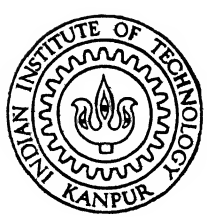
A GRAPHICAL QUERY LANGUAGE FOR EXTENDED ENTITY RELATIONSHIP MODEL

by

SANJAY JAIN



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1990

A GRAPHICAL QUERY LANGUAGE FOR EXTENDED ENTITY RELATIONSHIP MODEL

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by
SANJAY JAIN

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
FEBRUARY, 1990

Certificate

This is to certify that the thesis **A GRAPHICAL QUERY LANGUAGE FOR EXTENDED ENTITY RELATIONSHIP MODEL** has been carried out by Mr. Sanjay Jain under my supervision and the same has not been submitted elsewhere for the award of a degree.

T V Prabhakar

(Dr. T. V. Prabhakar)
Assistant Professor,
Computer Sc. and Engg. Dept.
Indian Institute of Technology,
Kanpur-208016.

Feb 1990.

Acknowledgments

I express my heartfelt thanks to Dr. T. V. Prabhakar for his expert guidance and encouragement during his supervision of this thesis work. His suggestions and critical remarks were very much helpful during the development and the preparation of this thesis.

I am thankful to C S E Lab. staff for their assistance and kind co-operation during this work.

I would also like to thank my fellow-students A. Chawla and S. N. Murty for their suggestions and support during the preparation of this thesis.

Lastly, but not the least, I thank all those who are either directly or indirectly involved in carrying out this thesis work.

Abstract

This thesis presents a scheme for handling graphical queries over an Extended Entity Relationship (EER) model. A user friendly graphics interface is built to define the database schema in the form of an EER diagram. The physical storage of data is in the relational model of the ORACLE RDBMS. The EER diagram is converted to equivalent relations.

A Graphical Query Language, which is easy to understand and can be used by persons without a prior knowledge of database systems, has been defined over the EER model. The syntax of the GQL is defined by a set of graphical operators. A GQL query is then converted to an SQL query and executed over the ORACLE RDBMS. The main features of the GQL are: Incremental query facility, browsing and consistent graphical update operations. GQL is shown to be at least as powerful as relational algebra.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Introduction

Database Management systems are used to store data in an organized way. There are three basic data models: Relational model, Network model, and Hierarchical model. Each of them has some advantages and disadvantages. Various query languages are provided on each model.

While designing a database schema for an application, we normally tend to design it at the conceptual level. The design is then realized in the database by using either of the above models.

A query language is the most important part of any database system. However, if the query language is over a low level data model (i.e. Relational/Hierarchical/Network etc.) then it becomes difficult to formulate queries. This is because a user thinks of the query at conceptual level and then tries to convert the query to the physical data model he has chosen.

This thesis is an attempt to bridge the gap between the query at conceptual level and query at the data model level. It presents a scheme for handling Graphical queries over the Extended Entity Relationship model.

We have selected "Extended Entity Relationship" model for designing the database schema. The physical storage of data is in

the Relational model of ORACLE RDBMS. A graphical interface is provided to the user to formulate his queries. These queries are then converted to SQL queries.

A user friendly interface, which specially benefits the naive users, is provided during designing of schema, as well as at the time of querying.

1.2 Features of the Graphical Query Language (GQL)

Following are the features of our GQL:

1) **Schema Presentation:** The Graphical Query Language is defined over an EER diagram. Since the database schema is also defined over an EER diagram, a query can be formulated and understood easily.

2) **Incremental Query:** It also provides the ability to construct new queries based on the result of existing ones.

3) **Completeness:** The GQL has been proved to be atleast as powerful as the relational algebra.

4) **Browsing:** Browsing of an entity or a relationship of any degree is possible.

5) **Consistency:** Graphical updating of data ensures data consistency. If an inconsistent operation is attempted then error is reported.

In addition to this a user friendly Graphics editor (specially for dealing with EER diagrams) is provided, which is used for designing and querying on the schema.

1.3 Thesis Layout

In this thesis, Chapter 2 discusses the Extended Entity Relationship model and the Graphical representation of the model. It also discusses the normal form of an EER diagram. A sample database schema used in this thesis is also discussed there. Chapter 3 discusses the algorithm for translation of EER diagram to relations.

In chapter 4, section 1 deals with the syntax of GQL and its operators. Various examples are taken to show different kinds of queries in GQL. Section 2 describes an algorithm to convert Graphical queries into equivalent SQL queries. Section 3 shows that GQL is as powerful as the relational algebra. Section 4 discusses the algorithm which ensures the consistency of the data. Section 5 discusses the methods of browsing the entities and relationships in the database.

Chapter 5 gives the implementation details of the GQL and finally, chapter 6 presents the conclusion and future work that can be done.

1.4 Architecture

Architecture of the thesis is as shown in Fig. 1.1. The user interacts with the graphics editor. Graphics editor interacts with the schema handler, Query handler, Browser, and Update module. Schema handler is used to design the database schema.

Query handler is used to formulate queries and execute them. Browser allows the user to examine the structure of the database. Update module is used to update the database information with the consistency checks on data.

All the above four module interacts with the ORACLE RDBMS.

1.5 Related Work

The conceptual design of databases has attracted lot of attention. Starting with Entity Relationship model [1], several conceptual database models have been proposed. Various query languages have also been also proposed on these models.

"TAXIS"[13] conceptual database system deals with three kinds of classes: Data class, Transaction class and the Exception class. "IRIS"[14] is a directed graph data model. It consists of "Objects", "Types", and "Functions". "Objects" represent things and concepts, "Types" represent a set of objects that share common properties, and "Functions" represent properties of objects. "Semantic Nets"[15] data model consists of "nodes" and "labels". It has four different kinds of nodes namely (i) Concepts, (ii) Events, (iii) Characteristics and (iv) values. Associations between nodes have "labels". The other data model proposed are "SIM"[17] and "Nested Datalog"[18]. "SIM" deals with two types of classes: Superclasses and Subclasses. "Nested Datalog" consists of nested predicates which define a tuple or a set constructor.

A lot of work has also been done on the Entity Relationship model. A normal form for the EER diagram is defined by Tok-wang Ling[12]. Category abstraction[11] (specialization and Generalization) is also introduced in the Entity Relationship model to capture real world semantics. Various kinds of attribute mappings to describe an Entity or a Relationship have been proposed.

Various graphical query languages on ER model have also been proposed. Azmoodeh M. and Hongbo D.[7] proposed a graphical query language which provides a set of pattern images and a set of simple rules for constructing pattern graphs. Users describe their queries by drawing pattern graphs against the graphical schema of the database.

A GQL proposed by Zhang Z. and Mendelzon A. O.[4] applies the concept of maximal objects from relational database theory to find a default connection among any set of nodes (selected for querying) in a database diagram. A graphical query facility proposed by Elmasri R. A. and Larson J. A.[8], rephrases a query using natural language and is presented to the user for verification.

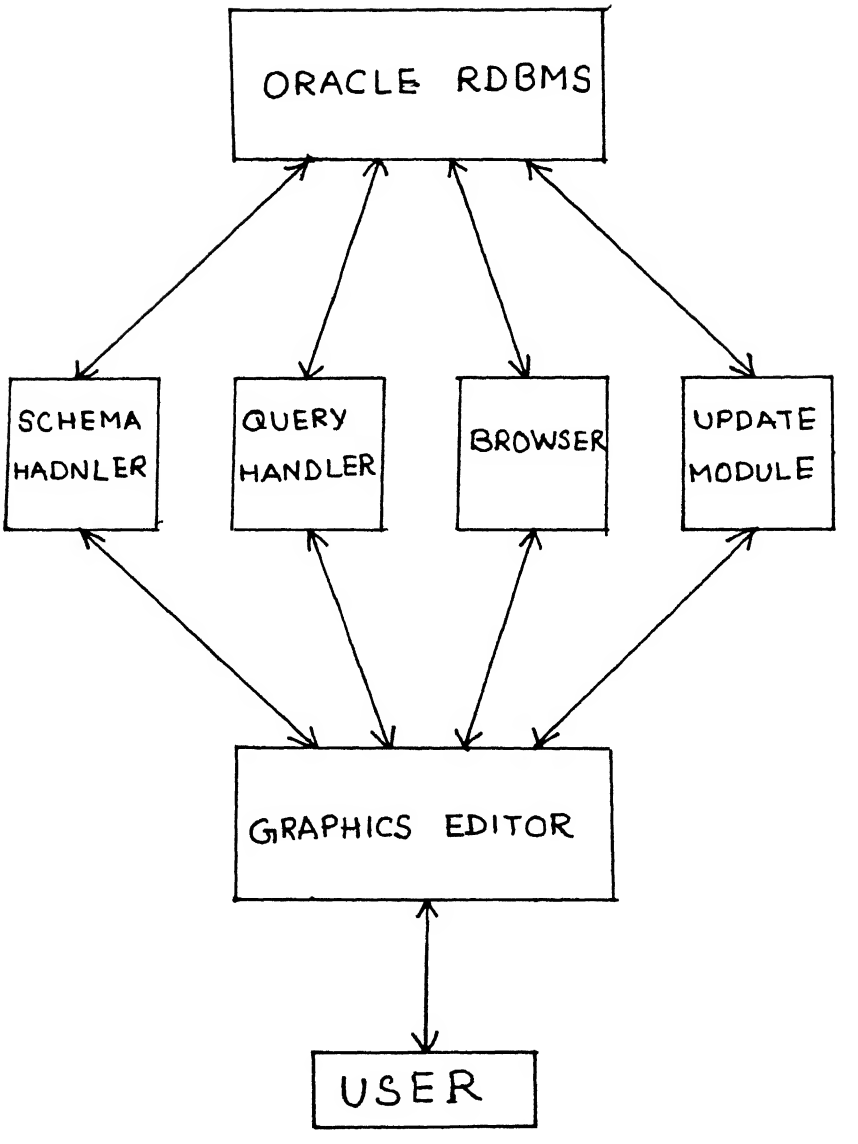


Fig. 1.1. Architecture

Chapter 2

Extended Entity-Relationship model

Entity Relationship (ER) model was initially proposed by Chen[1] and later modified and extended by many others. The Extended Entity Relationship (EER) model is based on a perception of the real world.

2.1 Entity Relationship Model

It consists of three classes of objects: Entities, Relationships and Attributes. We describe below the related definitions and concepts.

2.1.1 Entity:

An Entity is a distinct object. For example, a part, a supplier, a house are entities. An entity set is a set of entities such that each entity satisfies a predefined set of common properties. Examples of entity sets are, all suppliers, all houses, all parts etc.

2.1.2 Relationship:

A Relationship is an association among several entities. For example, in Fig. 2.1, "SUPPLY" is a relationship between the entities "PART" and "SUPLIER".

Degree of a relationship: The degree of a relationship is the total number of entities associated with the relationship. For example in Fig. 2.1, relationship "SUPPLY" is of degree 2 and in

Fig. 2.2, relationship "OFFER" is of degree 3.

Connectivity of a Relationship: The Connectivity of a Relationship specifies the mapping of the associated entity occurrences in the relationship. A connectivity could be either "one" or "many". For a relationship R, associated with entities E_1, E_2, . . . ,E_i, . . . ,E_n. A connectivity of "one" to E_i means that given all entity instances except E_i, participating in R, there is at most one related entity occurrence of E_i.

In Fig. 2.2, relationship "OFFER" is defined as follows: A teacher can teach one course in a department but can teach in different departments. A course offered by one department can be taught by only one teacher but same course may be offered in many departments. Entities "COURSE" and "TEACHER" are having a connectivity of "one", whereas entity "DEPT" has connectivity of "many".

2.1.3 Attributes:

Both entities and relationships have attributes representing their structural properties. Usually the domain for an attribute will be a set of integers, real numbers, or a character string.

There are two types of attributes:

1) **Key:** Set of attributes whose value uniquely identifies each entity or relationship is called the key for that entity or relationship. For example attribute "SNO" is a key for entity "SUPLIER" (Fig. 2.1). Since supplier number (SNO) uniquely identifies a supplier.

2) Descriptor: These attributes are used to describe an entity or relationship. For example attribute "SNAME" of entity "SUPLIER" in Fig. 2.1 describes the name of the supplier.

A descriptor attribute can have four types of mapping on to an entity or a relationship. These mapping are not defined in the ER model proposed by Chen[1].

(i) One to one: A "one to one" attribute uniquely determines the key i.e. for each value of that attribute there exists a unique value for key. For example attribute "SSN" (social security number) in entity "SUPLIER" is a "one to one" attribute. Since each citizen has a unique social security number.

(ii) Many to one: A "many to one" attribute can have same value for more than one key value. For example attribute "SNAME" (supplier name) of entity "SUPLIER" is a "many to one" attribute, since many suppliers may have the same name.

"One to one" or "many to one" attributes are also called as single valued attributes.

(iii) One to many: A "one to many" attribute has a set of values associated with each key value. But each value of "one to many" attribute uniquely determines the key value. For example attribute "CHILD" of the entity "SUPLIER" is a "one to many" attribute, since each supplier may have more than one child and a child can not have two fathers (Assuming that all suppliers are male).

(iv) Many to many: A "many to many" attribute has a set of values associated with each key value and each value of this

attribute may be associated with many key values. For example attribute "QUALIF" (qualification) is a "many to many" attribute, since a supplier may have many qualifications and a qualification may be associated with many suppliers.

2.2 Generalization and Specialization

The introduction of category abstraction into the ER model resulted in two additional types of objects: Specialization and Generalization.

2.2.1 Specialization:

An entity E_1 is a "specialization" of entity E_2, if every occurrence of E_1 is also an occurrence of E_2. A specialization entity will inherit all the attributes of its parent entity. It can also have additional descriptor attributes. If entities E_1, E_2, ... , E_m are the specialization of entity E then

all occurrences of E_i $\cap$ all occurrences of E_j, $i \neq j$

need not be a null set.

For example in Fig. 2.1, the entities "RAREPRT" and "BIG_PRT" are specialization of entity "PART". A part could be a rare part and also a big part.

2.2.2 Generalization:

An entity E is a generalization of the entities E_1, E_2, ... , E_n if each occurrence of E is also an occurrence of one and only one of the entities E_1, E_2, ... , E_n.

Generalization is used to emphasize the similarities among

lower-level entity types. An entity E at a lower level of generalization tree will inherit all the attributes from entities at higher levels. In addition it can also have its own descriptor attributes. For example entity "SUPLIER" is a generalization over "supplier from India", "supplier from USA", "supplier form Pakistan" and "supplier from USSR".

The Graphical representation of an EER model is as shown in Fig. 2.3.

2.3 Functional Dependencies in an EER diagram

Let primary-key(E) denote the set of attributes which form the primary key for an entity E. Let a relationship R be associated with the entities E_1, E_2, ... , E_m. If an entity E_i has connectivity of "one" associated with relationship R then, following are the functional dependencies associated with R.

for x <> i $U_{x=1}^{x=m}$ primary-key(E_x) --> primary-key(E_i).

hence the key of a relationship R is defined as

for x <> i $U_{x=1}^{x=m}$ primary-key(E_x).

Number of keys in a relationship is defined by the number of entities having Connectivity of "one" for that relation. In case of a relationship with no entity having connectivity of "one", the number of keys is one and defined as below:

$U_{x=1}^{x=m}$ primary-key(E_x).

Let key-relation(R) denote the set of all key of a relationship R. If X is an entity or Relationship, then let

key(X) denote primary-key(X) if X is an entity, or key-relation(X) if X is a relationship.

For each descriptor attribute 'A' of X, following are the functional dependencies defined for different mapping of 'A':

(i) One to One

key(X) --> A

A --> key(X)

(ii) many to one

key(X) --> A

(iii) one to many

key(X) -->> A

A --> key(X)

(iv) many to many

key(X) -->> A.

2.4 Normal form of an EER diagram

A normal form of an EER diagram was defined initially by Chung and later extended by Ling[12]. Following are the objectives of defining normal forms of an EER diagram:

(i) To capture and preserve all the semantics of the real world.

(ii) To ensure that all the relationships represented in EER diagram are non-redundant.

(iii) To ensure that all the relations translated from the EER diagram are either in 3NF or 5NF. (Proof is given in the [12]).

An EER diagram D is in normal form (ER-NF) if it satisfies following condition:

(i) All attribute names are distinct and of different semantics. For example in Fig. 2.4(a), attribute "CHILD_NO" is playing same role in two different entities "PERSON" and "CHILDREN". Therefore it should be modeled as in Fig. 2.4(b).

(ii) All functional dependencies should be defined by an EER diagram. There should not exist any other functional dependencies. For example in Fig. 2.5(a), the functional dependencies defined by it are:

ENO --> CITY

ENO --> NAME

ENO --> PIN

But an additional functional dependency (PIN --> CITY) also exists. Hence the above schema is not in ER-NF. The above schema can be modeled as in Fig. 2.5(b).

(iii) A relationship R should not be implied by any other relationships. For example in Fig. 2.6, relationship "MOTHER" is implied by the relationships "FATHER" and "WEDS". Hence relationship "MOTHER" is a redundant relationship and above schema is not in ER-NF.

2.5 Sample Schema

Let us take a sample database schema as shown if Fig. 2.1. We will now discuss the schema in detail.

Entity "PART" is a set of all parts.

Attribute "PNO" denotes part number.

Attribute "PNAME" denotes name of the part.

Attribute "COLOR" denotes color of the part.

Attribute "WEIGHT" denotes the weight of the part

Entity "SUPLIER" is a set of all supplier.

Attribute "SNO" denotes supplier number.

Attribute "SNAME" denotes name of the supplier.

Attribute "SSN" denotes the social security number.

Attribute "CHILD" denotes the children of the supplier.

Attribute "QUALIF" denotes the qualifications.

Entity "STORE" is a set of all stores.

Attribute "STNO" denotes the store number.

Attribute "LOC" denotes the location of the store.

Attribute "CAP" denotes the capacity of the store.

Entity "BIG_PRT" is a set of all big parts from entity "PART".

Attribute "SIZEP" denotes the size of the big part.

Entity "RAREPRT" is a set all rare parts from entity "PART".

Attribute "COST" denotes the cost of the rare part.

Entity "IND_SUP" is a set of all suppliers from India.

Attribute "STATE" denotes the state of the Indian supplier.

Attribute "CITY" denotes the city of the Indian supplier.

Entity "USA_SUP" is a set of all suppliers from U.S.A.

Attribute "PARTY" denotes the party of the U.S.A. supplier.

Entity "PAK_SUP" is a set of all suppliers from Pakistan.

Entity "USR_SUP" is a set of all suppliers from U.S.S.R.

Attribute "TEL_NO" denotes the telephone number.

Attribute "STATUS" denotes the status.

Entity "NORTH" is a set of all U.S.A. suppliers from north.

Entity "SOUTH" is a set of all U.S.A. suppliers from south.

Attribute "TEL_NO" denotes the telephone number.

Relationship "SUPPLY" means that a supplier supplies a part.

Attribute "QTY" denotes the quantity of the part supplied.

Attribute "CODE" denotes the code number.

Attribute "M_MAN" denotes the middle men involved.

Relationship "ALT"(alternate) means that a rare part has a alternate part.

Relationship "KEPT" means that parts are kept in a store.

Relationship "OWN" means that stores are owned by U.S.S.R. suppliers.

Relationship "GL"(group leader) means that indian suppliers are group leaders of the group of suppliers.

Entity "USA_SUP" is a generalization over entities "NORTH" and "SOUTH" defined by the relationship "ZONE". Entity "SUPLIER" is a generalization over entities "IND_SUP", "USA_SUP", "PAK_SUP", "USR_SUP" defined by the relationship "COUNTRY".

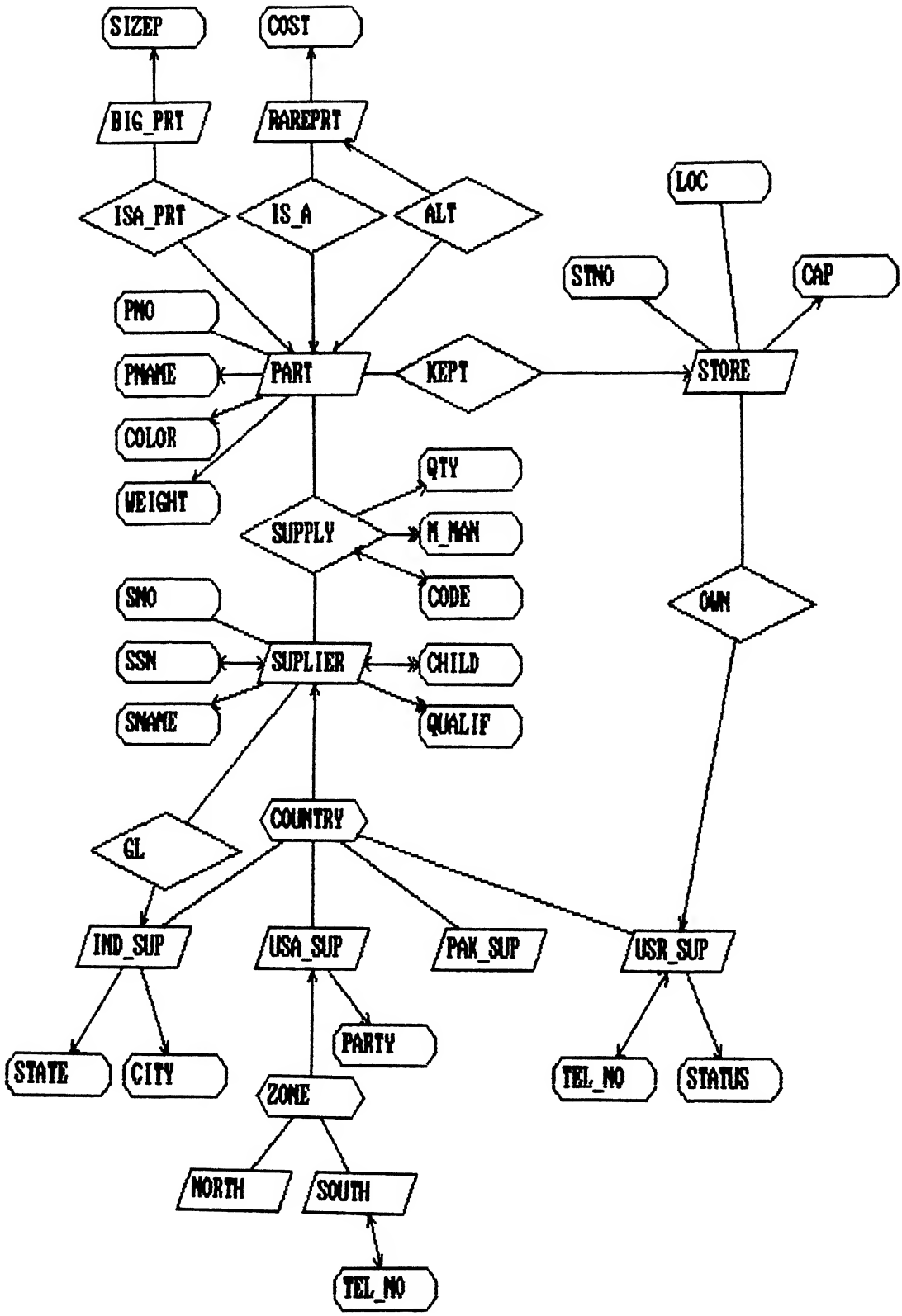


Fig. 2.1 Sample schema.

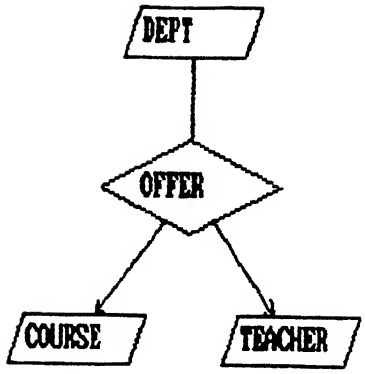
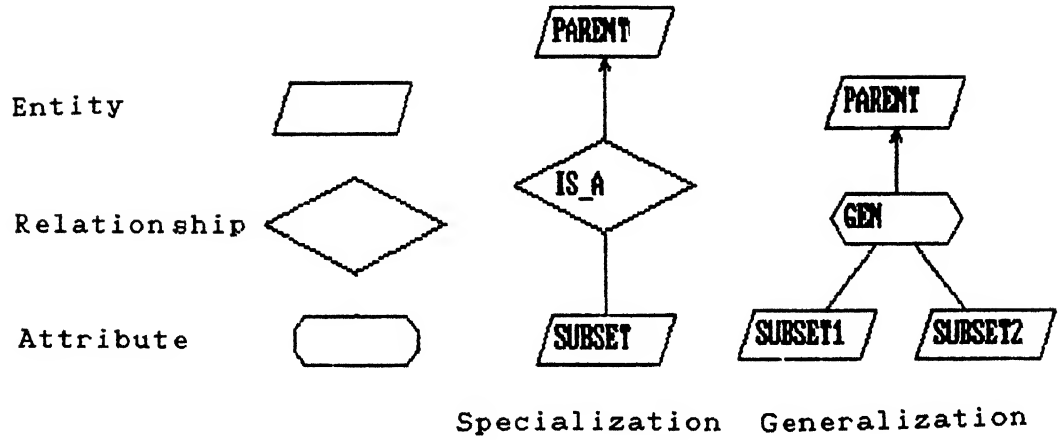


Fig. 2.2 Example of a relationship.



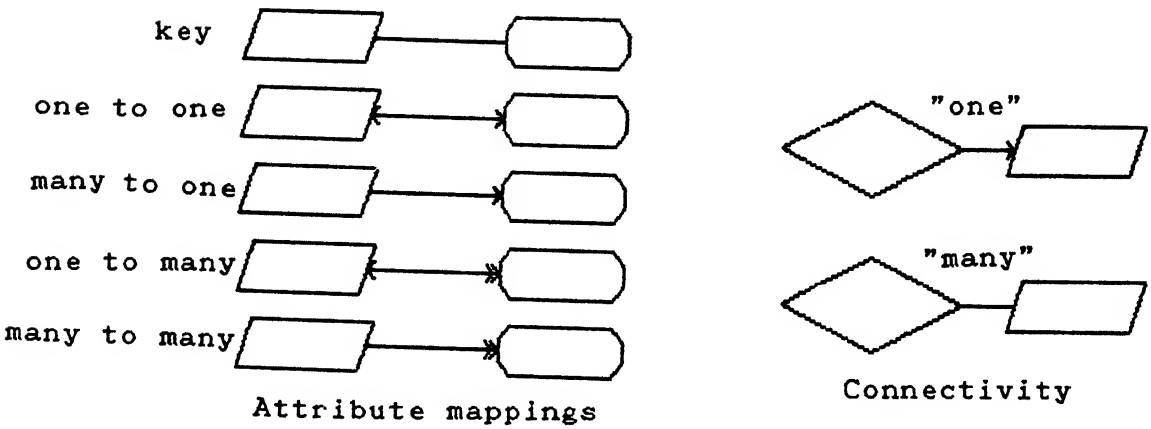


Fig. 2.3 Graphical representation of EER model.

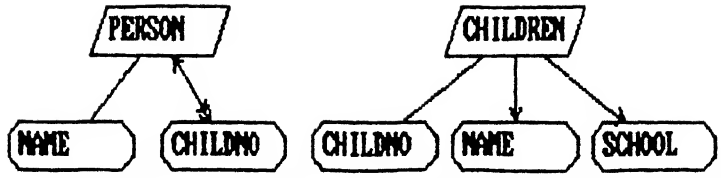


Fig. 2.4(a) It shows that attribute "CHILDNO" is playing same role in two different entities. Hence diagram not in ER-NF.

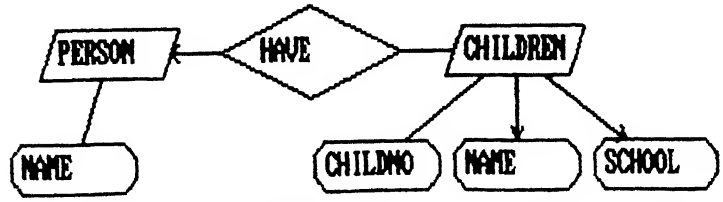


Fig. 2.4(b) It shows an alternate way to model EER diagram of Fig 2.4(a). Also it is in ER-NF.

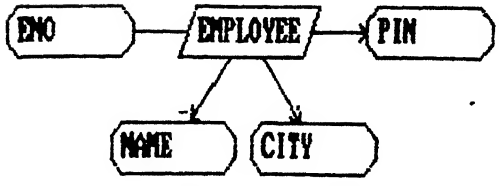


Fig. 2.5(a) It shows an Entity which has attributes such that all FDs are not defined by it.

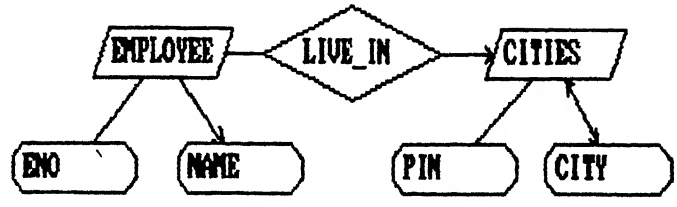


Fig. 2.5(b) It shows equivalent of the Fig. 2.5(b), with all the FDs defined by it.

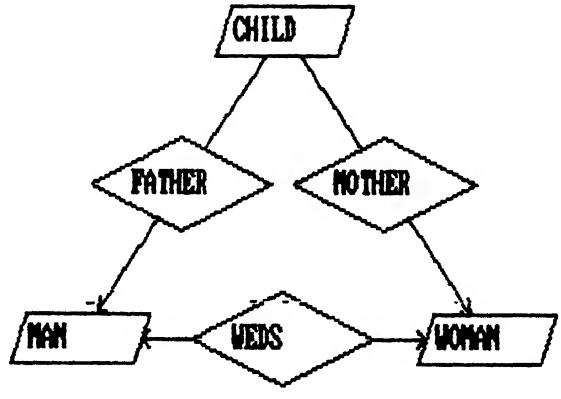


Fig. 2.6 It shows an EER diagram having redundant relationship.

Chapter 3

Translation of an EER diagram to Relations

Once the database schema has been defined by the EER diagram, it has to be realized by the physical data model. For each entity and a relationship defined in EER diagram, there will be one or more tables generated. We call relations (of Relational model) as tables to avoid confusion. In this section, an algorithm is given to translate an EER diagram to a set of tables. If the EER diagram is in normal form then tables generated are in 3NF or 5NF. The algorithm works in two steps:

Step 1: Translate each entity to a table.

(i) All the key attributes, "one-to-one" attributes, and "many-to-one" attributes form a table. Let us call this table an E-1 type table.

(ii) Each "one-to-many" and "many-to-many" attribute, along with the key attributes forms a table. Let us call these tables as E-2 type tables.

Step 2: Translate each relationship to a table.

(i) All the key attributes of the entities participating in the relationship R and all "one-to-one" and "many-to-one" attributes of R form a table. Let us call this table as R-1 type table.

(ii) Each "many-to-many" and "one-to-many" attribute of R along with any one key of relationship R form a table. Let us call this table as R-2 type table.

Whenever an entity or relationship has more than one table then a unique name to other tables are given by suffixing a "$" followed by an integer number. Similarly when two columns of a table have the same name then again one of them is suffixed by a "$".

For the EER diagram shown in Fig. 2.1, the table generated for all entities and relationships are as shown below:

```
Entity SUPLIER
1) SUPLIER
   1) SNO
   2) SSN
   3) SNAME
2) SUPLIER$1
   1) SNO
   2) CHILD
3) SUPLIER$2
   1) SNO
   2) QUALIF

Entity PART
1) PART
   1) PNO
   2) PNAME
   3) COLOR
   4) WEIGHT

Entity IND_SUP
1) IND_SUP
   1) SNO
   2) STATE
   3) CITY

Entity USA_SUP
1) USA_SUP
   1) SNO
   2) PARTY

Entity PAK_SUP
1) PAK_SUP
   1) SNO

Entity USR_SUP
```

```
1) USR_SUP
   1) SNO
   2) TEL_NO
   3) STATUS

Entity NORTH
1) NORTH
   1) SNO

Entity SOUTH
1) SOUTH
   1) SNO
   2) TEL_NO

Entity BIG_PRT
1) BIG_PRT
   1) PNO
   2) SIZEP

Entity RAREPRT
1) RAREPRT
   1) PNO
   2) COST

Entity STORE
1) STORE
   1) STNO
   2) LOC
   3) CAP


Relationship SUPPLY
1) SUPPLY
   1) SNO
   2) PNO
   3) QTY
   4) CODE
2) SUPPLY$1
   1) SNO
   2) PNO
   3) M_MAN

Relationship GL
1) GL
   1) SNO     /* IND_SUP */
   2) SNO$1   /* SUPLIER */

Relationship ALT
1) ALT
   1) PNO      /* RAREPRT */
   2) PNO$1    /* PART    */
```

```
Relationship OWN
1) OWN
    1) STNO
    2) LOC
    3) SNO

Relationship KEPT
1) KEPT
    1) PNO
    2) STNO
    3) LOC
```

Chapter 4

Graphical Query Language

The query interface is the most important part of any data base system. As the use of computers increases, many non-technical persons will need to interact with the query interface. A Graphical Query Language is defined over the EER model which is easy to understand and can be used by persons without a strong knowledge of database systems. This Graphical Query Language (GQL) is atleast as powerful as relational algebra.

4.1 Syntax of the GQL

In this section we will discuss the syntax of the GQL. A Graphical Query syntax is defined over five different graphical operators:

1) Mark (α)
2) Set Condition (β)
3) Delete (τ)
4) Duplicate (δ)
5) Naming (μ)

To understand the meaning of these operators let us take an example of an EER diagram as shown in Fig. 4.1.

Let an EER diagram be represented by D. Let Dq be the part of EER diagram used for querying. All the five operator defined above are over Dq. Initially Dq is equal to D.

4.1.1 Mark

This operator is used to mark the attributes which are of interest to the user. These are the attributes which are extracted from the database as a result of the query execution. Marked Attribute names are displayed in inverse video. For example, if we mark the attributes "PNO" and "PNAME" as shown in Fig. 4.2, it means that "show all part numbers and their names which are supplied by any supplier".

Note that this query is different from the query "show all part numbers and their names". This is because in Fig. 4.2 Relationship "SUPPLY" is also participating in the query. All the relationships present in the query participate to form the interpretation of the Query.

The unmarked attributes do not participate in the query. They can be optionally deleted without effecting the result of the query. Now consider another query: "to show all the suppliers and the parts supplied by the supplier along with the quantity of part supplied". This query is represented in GQL as shown in Fig. 4.3.

4.1.2 Set Condition

Queries involving conditions can be specified using the "set condition" operator. This operator can be used to give four different kinds of conditions.

(i) Conditions involving one attribute,

(ii) Conditions involving two attributes,

(iii) Logical Conditions (AND/OR/NOT),

(iv) Conditions for relating two queries (MINUS/UNION/INTERSECT)

Let us discuss each one in detail.

(i) Condition involving one attribute:

Any attribute of the diagram Dq can be selected for setting the condition. For example, consider the query "show all parts supplied by supplier S2". This query can be represented as shown in Fig. 4.4.

Any other condition on Dq will be ANDed with previous conditions. For example for the query "show all red parts supplied by supplier S2", the graphical representation will be as shown in Fig. 4.5.

(ii) Condition involving two attributes:

Any two attribute of Dq can be selected for this type of condition. These condition are of two types.

(a) Comparison operator

<, >, =, <>, <=, >=

(b) Nesting operator

ANY, ALL

Nesting operator is prefixed by a comparison operator.

In this subsection we will discuss the condition of type (a) only. Nesting operator will be discussed latter.

For the time being let us assume that supplier has an additional attribute called "WEIGHT". Although it is not shown in Fig. 4.1. Now if the query is: "show all suppliers who supply a

part such that supplier weight is less than the part weight". This query is represented as shown in Fig. 4.6. Direction of the comparison is shown by an arrow pointing to one attribute of the condition.

Suppose an additional condition on attribute "QTY" is added as shown in Fig. 4.7. The query of Fig. 4.7 represents "show all suppliers who supply a part such that supplier weight is less than part weight and quantity of that part supplied is greater than or equal to 20".

(iii) Logical condition (AND/OR/NOT): As shown in the previous example all the conditions on attributes are ANDed by default to form the query. But sometimes more complex queries are desired. For example, consider a query "show all parts which are either supplied by supplier S2 or supplied by any supplier having red color". This query is represented as shown in Fig. 4.8.

Logical conditions can be over single attribute conditions or over two attribute condition or over another logical condition. A more complex query is as shown in Fig. 4.9. This query means that "show all suppliers whose name is "JONES" and supplies some part OR a supplier who supplies a red part weighing less than 20".

"NOT" condition can be specified on a single/two attribute condition or on another logical condition. Consider the query shown in Fig. 4.10. This query means that "show all suppliers who supply a part of color other than red or green". Note that this query is totally different from the query that "show all

suppliers who do not supply a red or green part". Graphical representation of these type of queries will be discussed later.

(iv) Condition involving two queries: If two queries over the same set of domain return some tuples then final result of these two queries is formed by relating these two queries' result by any one of the following operators:

(a) UNION

(b) MINUS

(c) INTERSECT

Description of these operators is as shown in Fig. 4.11. Graphical queries using these operators are discussed later.

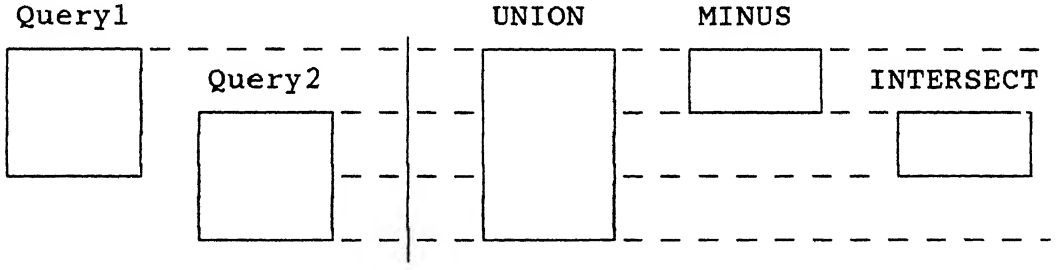


Fig. 4.11 Description of the operators used to relate two queries results.

4.1.3 Delete

In a query, most of the time we are not interested in the full EER diagram. Delete operator can be used for deleting the objects not used in the query. All the relationships present in the query participate to form the meaning of that query. Delete operator can be used for following purposes:

(i) Deleting a relationship.

(ii) Deleting a link from generalization entity to subset entities.

(iii) Deleting an Entity.

(iv) Deleting an Attribute.

In first two cases the result of the query is effected, while case (iii) & (iv) are only used to take out all redundant objects which are not participating in the query. For example if the query is to "show all part number and their names". This query can be constructed by first deleting the relationship "SUPPLY" and then marking the attributes "PNO" and "PNAME". The graphical representation of this query is as shown in Fig. 4.12. Note the difference between the queries of Fig. 4.2 and Fig. 4.12.

Let us now take a slightly complex schema as shown in Fig. 2.1. Consider the query "show all Indian suppliers who supply a part kept in a store identified by location L1 and store number ST1". This query is constructed by first using a series of delete operators. All the relationships except "SUPPLY", "KEPT", and "COUNTRY" are deleted and then connections between relationship "COUNTRY" to entities "USR_SUP", "USA_SUP" and "PAK_SUP" are also deleted. Now conditions on attributes are added using the "Set condition" operator discussed earlier. Finally concerned attributes are marked for display of result. Entities which are not participating in the query like "BIG_PRT", "PAK_SUP" etc. can be deleted without effecting the query result. This query is shown in Fig. 4.13.

Fig. 4.14 shows the query: "show all suppliers who supply a part which is a big part and also a rare part".

4.1.4 Duplicate

Operator "Duplicate" is used to duplicate an entity or a relationship. When a relationship is duplicated then it should be connected to all the entities (may be duplicate entities) with which it was originally associated. When an entity or relationship is duplicated then its image is created and this image can be used for querying the database. Duplicated objects are suffixed by "#".

For example if a query is to "show all suppliers who supply at least one part supplied by supplier S2", the graphical representation will be as shown in Fig. 4.15.

4.1.5 Naming

Nesting of a query is always a powerful tool for a query language. In this EER model based Graphical Query Language, nesting is done by first naming a valid query. Naming a query means treating result of that query as an additional entity in the EER diagram. This query is given a unique name by the user and all the marked attributes of this query are now the attributes of this new entity. All attributes of this entity form the key.

Let us take an example. Suppose the query is to "find out all the suppliers who supply a part such that weight of that part

is greater than the weight of all parts supplied by supplier S2". This query can be broken up into two queries.

(i) Find all the parts supplied by supplier S2.

Let us name this query as S2_PARTS

(ii) Find all the suppliers who supply a part whose weight is greater than the weight of all S2_PARTS.

The first query is as shown in Fig. 4.16(a). Let this query be named as S2_PARTS and a new query be created using nesting operator "=ALL", as shown in Fig. 4.16(b).

Nesting operator "ALL" is applied to all the tuples of the query. It returns true if all tuples satisfy the condition prefixed to the nesting operator. Nesting operator "ANY" returns true if any of the tuples satisfy the condition prefixed to "ANY".

It is not always necessary that a named query should be used as a nesting query, but it can also be viewed as an entity. For example consider the query "Show all suppliers who supply at least one part supplied by the supplier S2". This query can be represented as in Fig. 4.16(c).

Similarly an entity can also be viewed as a query and nesting operators can be used to form a query. For example Fig. 4.17 shows the query representing "all suppliers who supply a part whose weight is greater than the weight of all big parts".

When a named query or an entity is viewed as a query (i.e. connected through nesting operators) a condition on that nested query can not be specified. For example the query of Fig. 4.18 is

not a valid query. This restriction has been placed for ease of implementation.

Now let us discuss the condition operator relating the result of two queries. This operator can be specified on two named queries or entities. For example, consider the query "show all parts which are big or rare". This query is represented as shown in Fig. 4.19. All the common attributes (i.e. having same name) are selected for display of result.

4.2 Translation of GQL query to SQL query

This section discusses the algorithm to translate a query in GQL to SQL. These SQL queries are executed by the Oracle RDBMS. The algorithm consists of the following steps:

1) Join all relationships

   For all the entities participating in the query do

   (a) Find all the relationships associated with this entity.

   (b) Join all R-1 type tables of the corresponding relationship using the key of the entity for join.

2) Join within relationship

   For all the relationships participating in the query do

   (a) Find all multivalued attributes on which there is a condition. This gives all R-2 type tables which are used in the query.

   (b) For each R-2 type table found in the above step do

join that table with the R-1 table, using key of the R-2 table for join.

3) Join within Entity

For all the entities participating in the query do

(a) Find all the attributes on which there exists a condition. This gives all the tables participating in the query.

(b) Join all the tables found in the above step using key of the entity to which they belong.

4) Join for specialization/generalization entities

(a) For all specialization entities participating in the query do

Find the parent entity and join it with its specialization entity using the key of the parent entity.

/* Selection for the table of the entities to be joined is done by following rule:

Find an attribute other than the key attribute on which there exists a condition.

If such attribute is found then

use the table to which this attribute belongs for joining.

else

use E-1 type table for join

*/

(b) For all generalization entities participating in the query do

Find the child entity participating in the query. Join it with the generalization entity using the key of the generalization entity.

/* Selection of tables for entities to be joined is done by the rule given above in step 4(b). */

5) Join entity with relationship

For all the entities participating in the query do

If it has an attribute other then a key attribute on which a condition exists or the entity is joined in step 4 then

Join the table in which the attribute exists to the R-1 type table of any relationship associated with that entity. Use the key of the entity to join.

6) Generate condition

For each root condition of the query do

generate SQL condition by traversing the condition tree. (Condition over key attribute of an entity is moved to any relationship associated with the entity. This is done for optimization of query)

/* Root condition is that condition, which is not in the hierarchy of any other condition. For example in Fig. 4.8 logical condition "OR" is a root condition while in Fig. 4.7, both ">=" and "<" are root conditions. */

Note: Whenever a duplicate object is participating in the query then an image of all the tables of the duplicate object is created in the "from" clause of the SQL statement.

attributes used in GQL over this duplicated objects are qualified by the image of the object.

Consider a query shown in Fig. 4.20. This query represents "show all supplier numbers, their children and qualifications, and the middle man associated with the supply of the big parts kept in a store ST1 located at L1".

Equivalent SQL query is

```
Select SUPLIER$1.SNO, SUPLIER$1.CHILD, SUPLIER$2.QUALIF,
SUPPLY$1.M_MAN
from SUPLIER, SUPLIER$1, SUPLIER$2, SUPPLY, SUPPLY$1, KEPT
where SUPPLY.PNO = KEPT.PNO AND /* step 1 */
      SUPPLY.PNO = SUPPLY$1.PNO AND /* step 2 */
      SUPPLY.SNO = SUPPLY$1.SNO AND /* step 2 */
      SUPLIER$1.SNO = SUPLIER$2.SNO AND /* step 3 */
      PART.PNO = BIG_PRT.PNO AND /* step 4 */
      PART.PNO = SUPPLY.PNO AND /* step 5 */
      STORE.LOC = 'L1' AND /* step 6 */
      STORE.STNO = 'ST1' /* step 6 */
```

SQL query for the Fig. 4.15 is as shown below:

```
select SUPPLY.SNO
from SUPPLY, SUPPLY SUPPLY#
where SUPPLY.PNO = SUPPLY#.PNO AND
      SUPPLY#.SNO = 'S2'
```

4.3 Completeness of GQL

In this section, we prove that GQL is as powerful as relational algebra. In other words, we prove that all queries that can be generated by relational algebra can also be generated by GQL.

Theorem: If E is a relational algebra expression, then there is a equivalent representation to E in GQL.

Proof:

We prove this by induction on the number of occurrences of operators in E.

Basis: Zero operators. Then E is a table. In GQL this is represented as follows: All the columns of table E have a attribute in EER diagram. Mark these attributes. If the table E represents a relationship (R) in EER diagram then, all the relationships except R are deleted. Otherwise, if the table E represents a entity then all relationships are deleted. This query represents the table E. For example, tables "SUPLIER$1" and "SUPPLY$1" can be represented by the graphical queries as shown in Fig. 4.21(a) and Fig. 4.21(b) respectively.

Induction: Assume that E has at least one operator, and that the theorem is true for expressions with fewer occurrences of operators than E has.

Case 1: E = E1 U E2.

By induction hypothesis we assume that there exists a query

Q1 and query Q2 in GQL representing E1 and E2 respectively. E can be represented in GQL as shown in Fig. 4.22.

Case 2: E = E1 - E2.

Applying induction hypothesis as in case 1, then E can be represented as shown in Fig. 4.23.

case 3: E = E1 x E2.

Let E1 and E2 be equivalent to Q1 and Q2 respectively in GQL as in case 1. Let E1 have attributes u_1, u_2, ... , u_k and E2 have v_1, v_2, ... , v_m. These attributes will also be associated with Q1 and Q2 respectively. E can be represented by marking all attributes of Q1 and Q2 as shown in Fig. 4.24.

Case 4: E = $\pi_{i1, i2, \ldots, ik}(E1)$.

Let E1 be equivalent to Q1 in GQL by induction hypothesis. E can be represented by marking the attributes i1, i2, ... , ik of Q1. Mathematically E is equivalent to $\alpha_{i1, i2, \ldots, ik}(Q1)$.

Case 5: E = $\sigma_F(E1)$.

Let E1 be equivalent to Q1 in GQL. Then E is equivalent to $\beta_F(Q1)$ in GQL.

Hence GQL is as powerful as relational algebra.

4.4 Graphical specification of update operations

In this section we will discuss the algorithm for the graphical update operations on the EER diagram. The most important feature about update operations is that they automatically propagate to preserve the semantic integrity of the

database.

For example, in Fig. 2.1, if we delete an instance of an entity "PART" then automatically all the "SUPPLY" and "KEPT" instances of that part will be deleted. Also if that part is a rare part or a big part then its occurrences from those entities are also deleted.

If an update operation results in the violation of the integrity of the data then that operation is not allowed.
The operation is performed by moving the cursor over the entity, relationship, or attribute concerned and then pressing the key to initiate and select the type of operation to be performed. There are three kinds of update operations which can be performed over EER diagram.

(i) Insert

(ii) Delete

(iii) Modify

4.4.1 Insert

Insert operation can be done for an entity or a relationship.

Insert into an entity : If the entity is a subset entity then additional attributes are inherited from the parent entity. The user is asked to enter the values of each attribute. Values for key attributes are mandatory while it is optional for descriptor attributes.

If the values entered for an attribute are inconsistent,

then an error message is flashed. Otherwise values are inserted in the appropriate tables. Consistency of the data in an EER diagram means that the functional dependencies defined by the EER diagram should not be violated. For example the value entered for the attribute "SSN" of entity "SUPLIER (Fig. 2.1) should be unique. Additional constraints to insert an instance for a subset entity of generalization or specialization are discussed below.

When a subset entity in a specialization hierarchy is to be inserted then, key value should exists in the parent of it. Like in Fig. 2.1 a big part P can not be inserted unless there exists a part P in entity "PART". In case of a generalization hierarchy, an insertion at subset entity can only be done if key value of that subset entity does not exist in the generalization entity. For example in Fig. 2.1, if a new Indian supplier S is to be inserted to entity "IND_SUP" then, there should not exists a supplier S in any of the subset entities.

An algorithm to insert an instance into an entity is give below.

```
insert_into_entity(E)
  If entity is strong
      insert_into_strong_entity(E)
  else if entity is a subset entity in specialization hierarchy
      insert_into_specialization_subset_entity(E)
  else
      insert_into_generalization_subset_entity(E)
```

```
insert_into_strong_entity(E)
  If key value already exists
      insert_multivalued_attribute(E)
  else
      insert_all_attribute(E)

insert_into_specialization_subset_entity(E)
  If key value does not exists in the parent of E
      return(ERROR)
  else if key value exists in E
      insert_multivalued_attribute(E)
  else
      insert_all_attribute(E)

insert_into_generalization_subset_entity(E)
  Find root entity(Er) in the generalization hierarchy(Fig. 4.25)
  if key value exists in Er
       if key value exists in E
           insert_multivalued_attribute(E)
       else
           return(ERROR)
  else if Er is a strong entity
      insert_all_attribute(E)
  else /* Er is a specialization subset entity */
      if key value does not exists in the parent of Er
```

```
            return(ERROR)
      else
            insert_all_attribute(E)

insert_all_attribute(E)
  For each one-to-one and one-to-many attribute of E
      if value of that attribute already exists for any key value
            return(ERROR).
  Insert all attribute values in all the concerned tables.

insert_multivalued_attribute(E)
  for each one-to-many attribute of E
      if value of that attribute already exists for any key value
          return(ERROR)
  for each many-to-many attribute of E
      if value of that attribute already exists for the key value
to be inserted
            return(ERROR)
  insert all multivalued attributes in the concerned tables.
```

Insert into an relationship: To insert into a relationship, values of key attributes of all entities associated with the relationship and values of the attributes belonging to that relationship are asked from the user interactively. Now, values of the attributes are checked for consistency using the following algorithm.

```
insert_into_relationship(R)
  if the key value already exists in R
      insert_multivalued_attribute(R)
  else
      for each entity(E) associated with R
          if values of key attributes of E do not exists in E
              return(ERROR)
          if connectivity of E is "one"
              if functional dependency for E is violated
                  return(ERROR)
  insert_all_attribute(R)
```

4.4.2 Delete

Delete operation can be done for an entity or a relationship.

Delete from an entity: Deleting an instance from an entity may result into deleting the instances from the relationship associated with it. If an entity lies in the generalization tree then it also results in deleting instances from parent as well as subset entities. Algorithm to delete from an entity is given below:

```
delete_instance_from_entity(E)
  if the instance identified by key value exists in E
      delete all instances of that key value from E
  else
      return(No more delete)
  for each relationship R associated with E
```

```
        delete instances of R in which key value of E occurs
    if E is a subset entity in generalization hierarchy
        delete_instance_from_entity(parent(E))
    else if E is a generalization entity
        find  a subset entity Es of E in which the key value  of  E
exists
        delete_instance_from_entity(Es)
```

Delete from a relationship: Deleting an instance from the relationship is simple. User is asked to give the values of each key attributes of all the entities associated with relationship. Algorithm to delete from a relationship is given below.

```
delete_instance_from_relationship(R)
    if the instance identified by the key value of R exists in R
        delete all instances of that key value from R
    else
        return(ERROR)
```

4.4.3 Modify

Modify operation is provided only on the attribute. A value of a key attribute can not be modified. Only the values of the descriptor attributes can be modified. When an attribute to be modified is a multivalued attribute then the old value of that attribute is asked along with the key. If the key exists (in case of single valued attribute) or the old value of that attribute exists for that key (in case of multivalued attribute) then the value of the attribute is modified. Otherwise an error is returned.

4.5 Browsing

Browsing allows a user to examine, at a conceptual level, the structure of the database. There are two kinds of browsing mechanisms possible.

(i) Browsing an entity.

(ii) Browsing a relationship.

4.5.1 Browsing an entity

To browse an entity, all the relationships from the EER diagram are deleted. Now, the entity to be browsed is selected. Only one entity can be browsed at a time. First occurrence of that entity is displayed. All the attributes which are present participate automatically in the browsing. The Fig. 4.26 shows an example of browsing. The entity occurrences can be scrolled by the key attributes of the entity for scrolling. If the entity has multivalued attributes then that attribute can be scrolled independently for the same value of the key attribute. In Fig. 4.26(b) attribute "CHILD" is scrolled.

4.5.2 Browsing a relationship

To browse a relationship, all the other relationships are deleted. Now the relationship to be browsed is selected. In case the relationship is associated with the subset entities as shown in Fig. 4.27, the key attributes should be added to the subset

entity. All the attributes present in the entities associated with the relationship and all the attributes of relationship automatically participate in the browsing. The multivalued attribute information can independently be scrolled within the object.

The information about the relationship can be scrolled by selecting the key attributes of one of the entities associated with the relationship. Relationships can be scrolled in three ways.

(i) If the connectivity between the entity and the relationship is "many" then that entity is scrolled within the values of the other entities, and relationship information will be scrolled accordingly.

(ii) If the connectivity is "one" then, the new set of values for the other entities has to be retrieved. Relationship information will be scrolled accordingly.

(iii) An entity associated with the relationship can also be scrolled independently. All the entity occurrences which are participating in the relationship are scrolled one by one. Information about the other entities participating is also retrieved.

Fig. 4.28 and Fig. 4.29 shows the example of browsing the relationships "SUPPLY" and "KEPT" respectively.

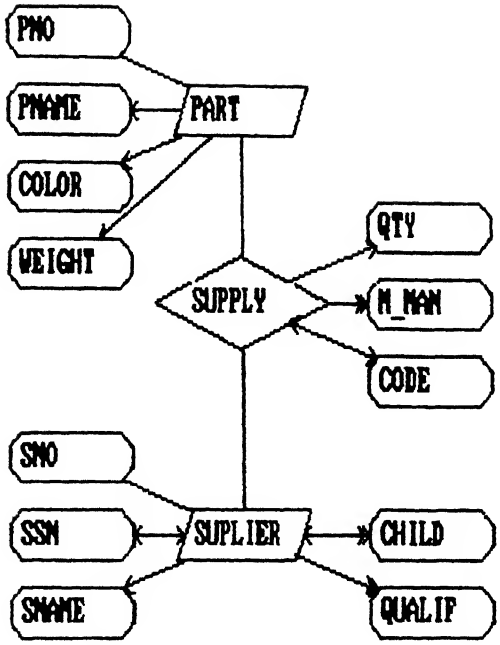


Fig. 4.1 Sample Schema 2.

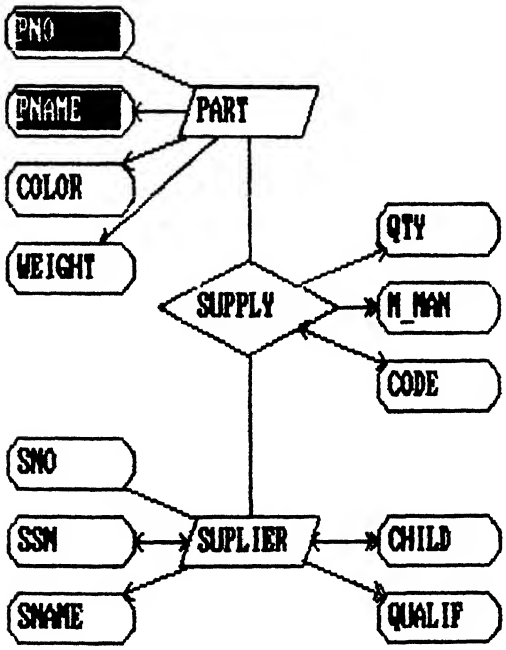


Fig. 4.2 Query: Show part number and name of all parts supplied.

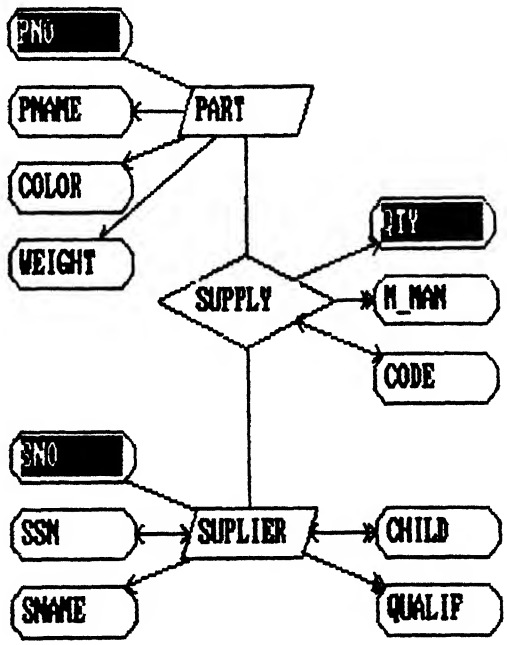


Fig. 4.3 Query: Show all suppliers and the parts supplied by the supplier along with the quantity of part supplied.

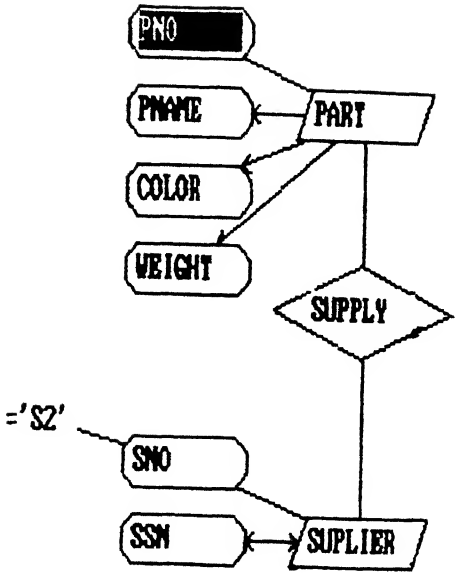


Fig. 4.4 Query: Show all parts supplied by supplier S2.

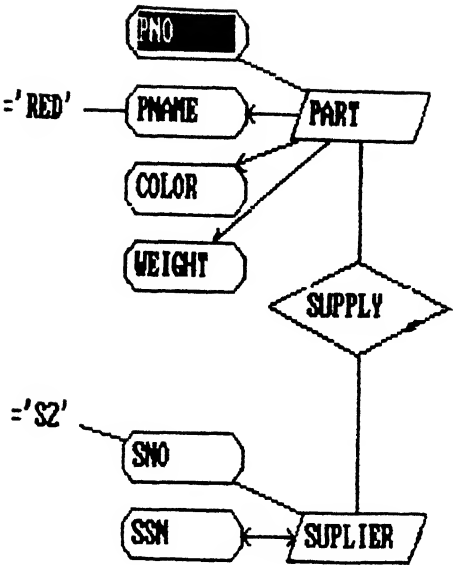


Fig. 4.5 Query: Show all red parts supplied by supplier S2.

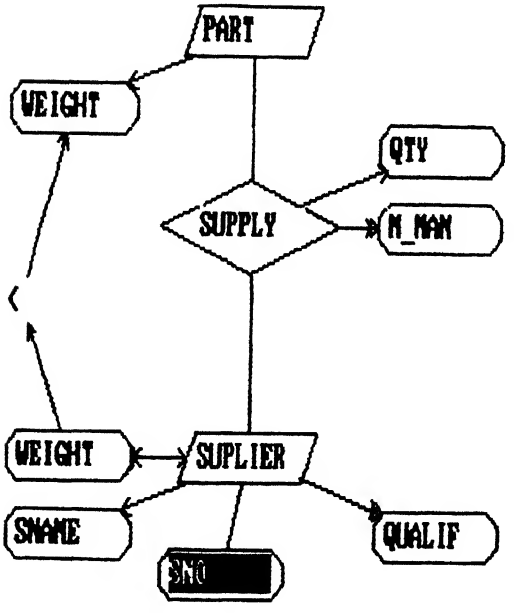


Fig. 4.6 Query: Show all suppliers who supply a part such that the supplier weight is less than the part weight.

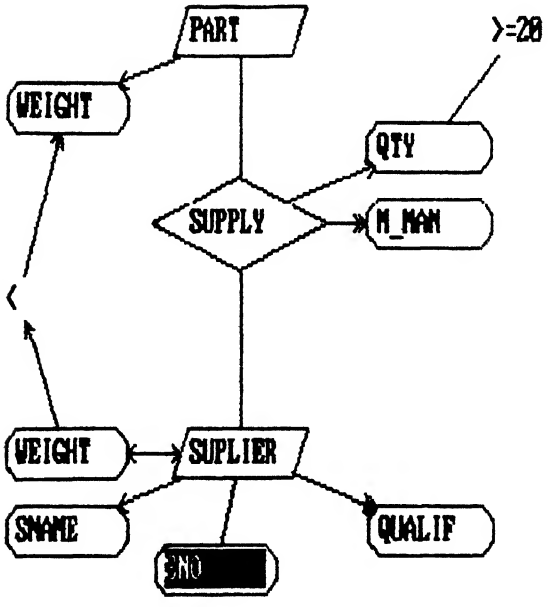


Fig. 4.7 Query: Show all suppliers who supply a part such that the supplier weight is less than the part weight and quantity of that part supplied is >= 20.

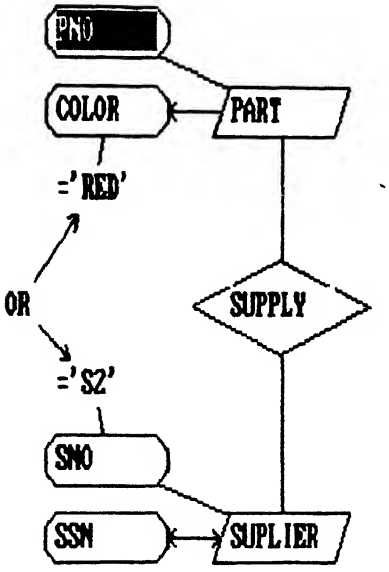


Fig. 4.8 Query: Show all parts which are either supplied by supplier S2 or supplied by any supplier having red color.

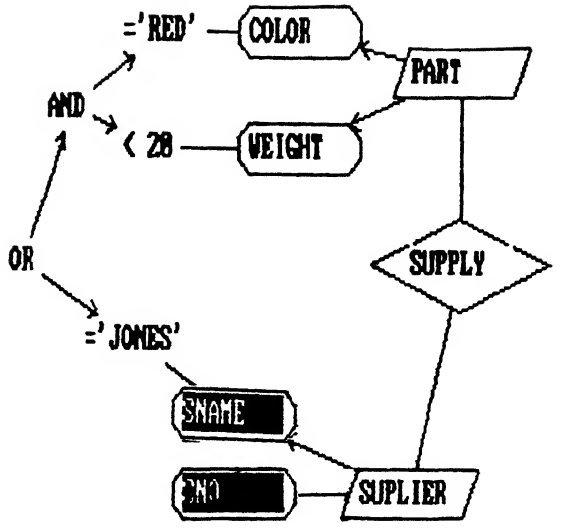


Fig. 4.9 Query: Show all suppliers whose name is JONES and supplies some part OR show all suppliers who supply a red part weighing less than 20.

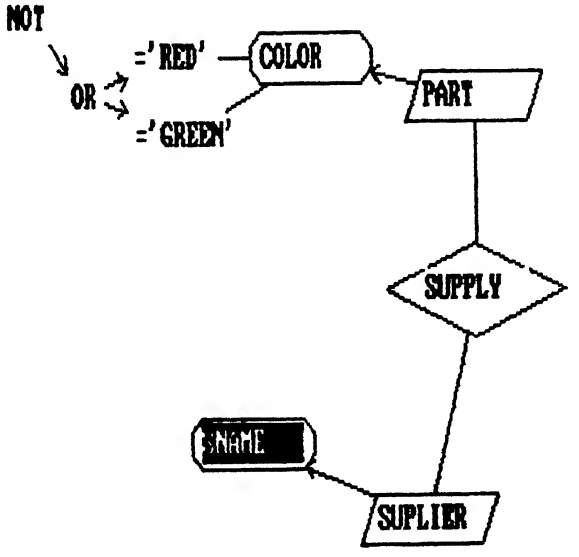


Fig 4.10 Query: Show all suppliers who supply a part of color other than red or green.

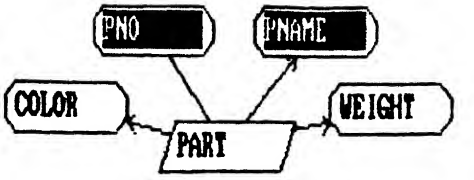


Fig 4.12 Query: Show all part numbers and their names.

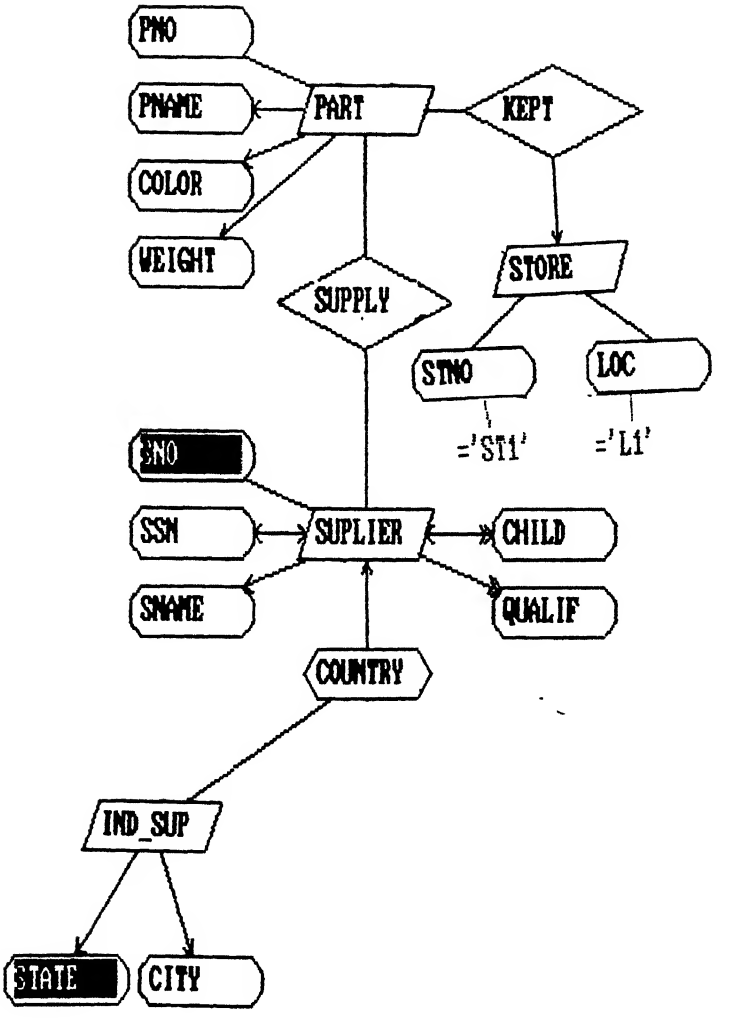


Fig 4.13 Query: Show all Indian suppliers who supply a part kept in a store identified by location L1 and store number ST1.

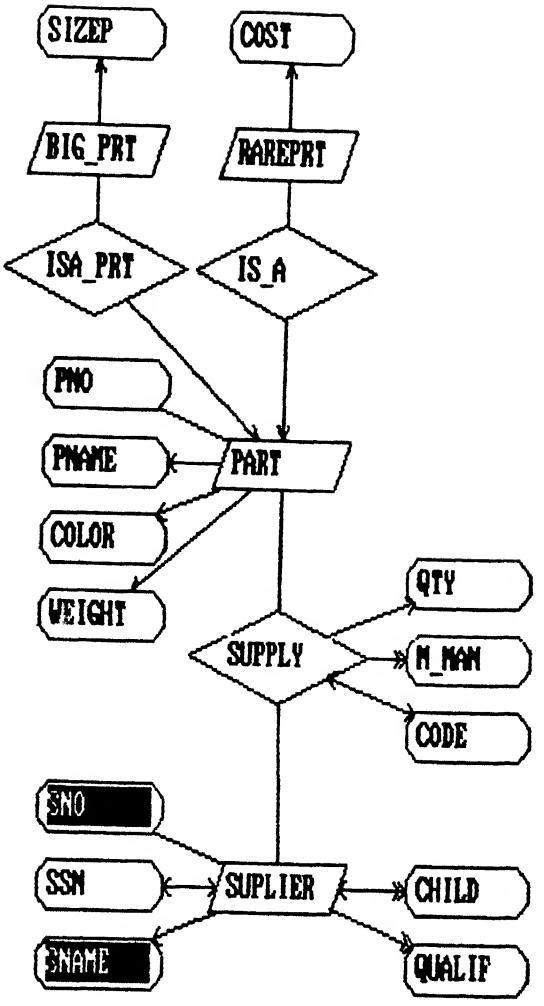


Fig 4.14 Query: Show all suppliers who supply a part which is a big part and also a rare part.

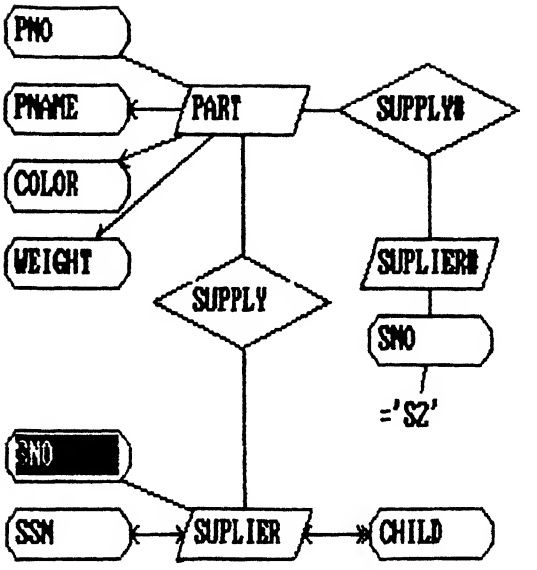


Fig 4.15 Query: Show all suppliers who supply at least one part supplied by supplier S2.

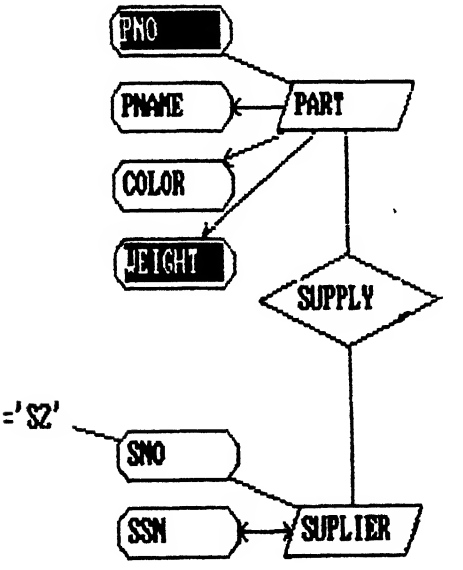


Fig 4.16(a) Query: Find all parts and the weight of the part supplied by supplier S2. This query is named as S2_PARTS.

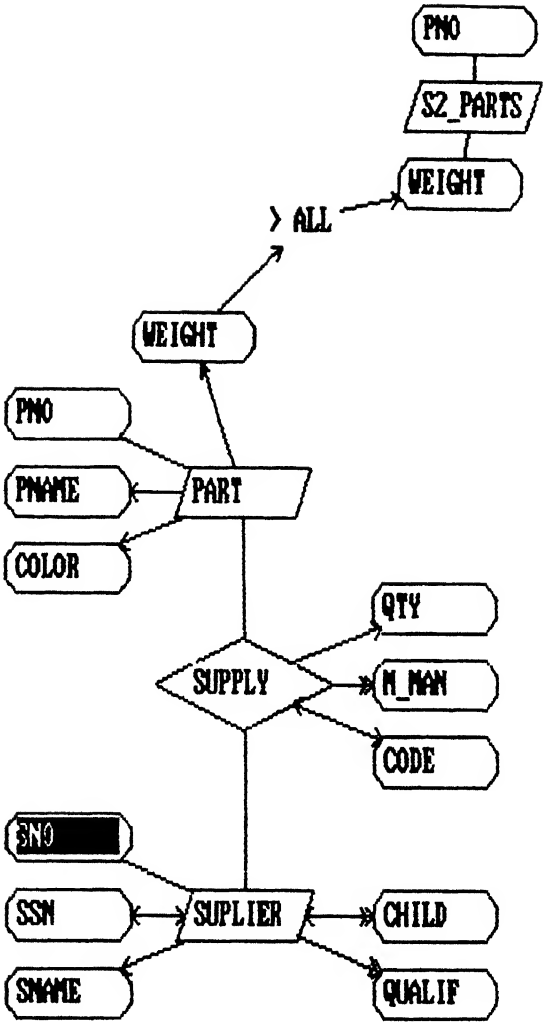


Fig 4.16(b) Query: Find all suppliers who supply a part whose weight is greater than the weight of all parts supplied by supplier S2.

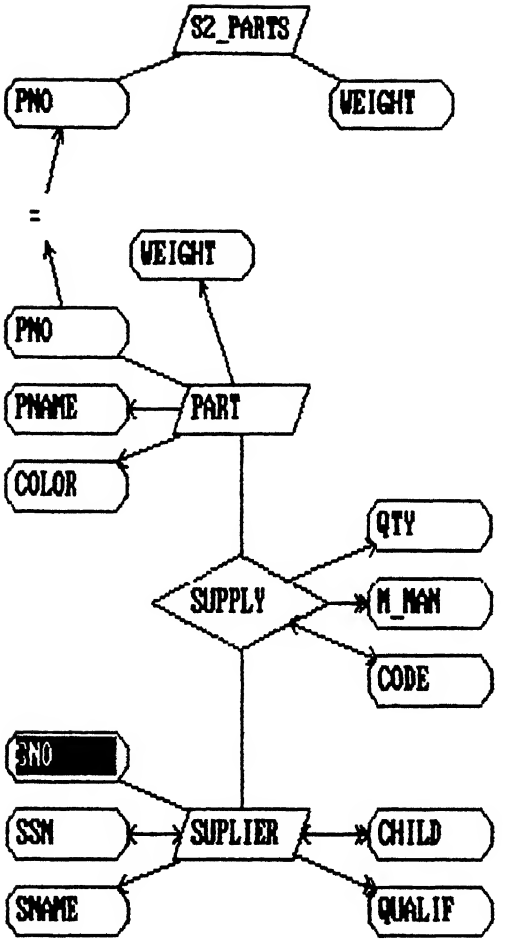


Fig 4.16(c) Query: Show all supplier who supply at least one part supplied by supplier S2. This query is same as shown in Fig 4.15.

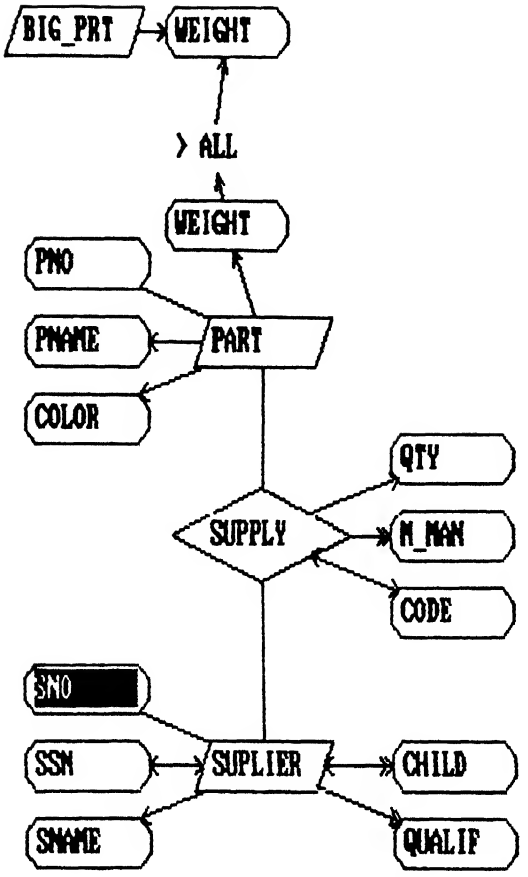


Fig 4.17 Query: Show all suppliers who supply a part whose weight is greater than the weight of all big parts.

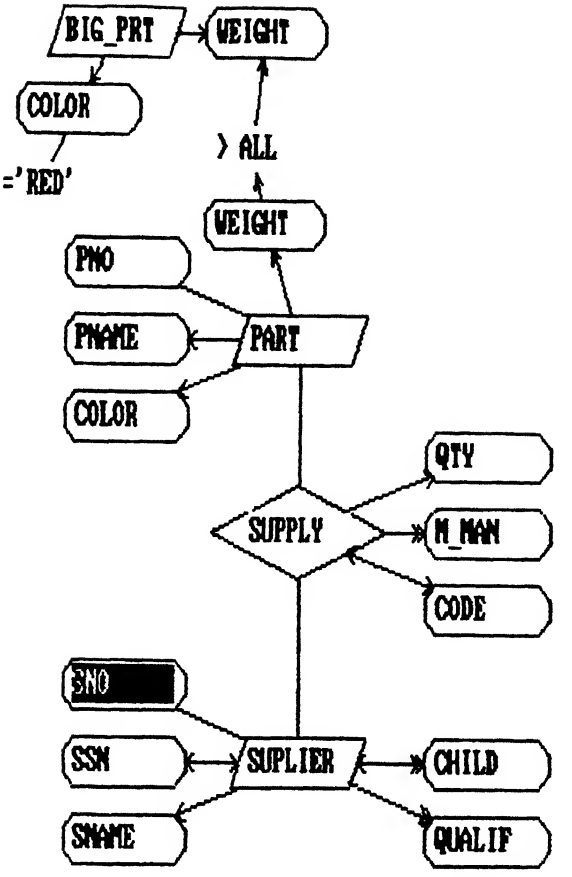


Fig 4.18 Invalid Representation of a query: Show all suppliers who supply a part whose weight is greater than the weight of all big red parts.

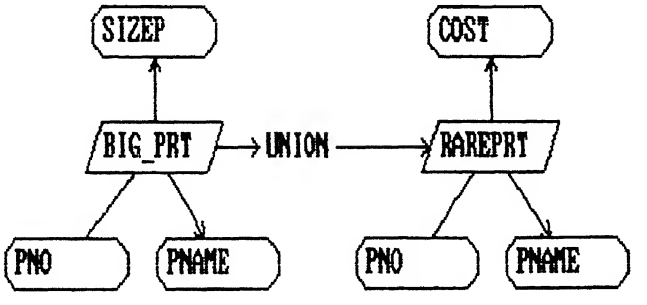


Fig 4.19 Query: Show all parts which are big or rare.

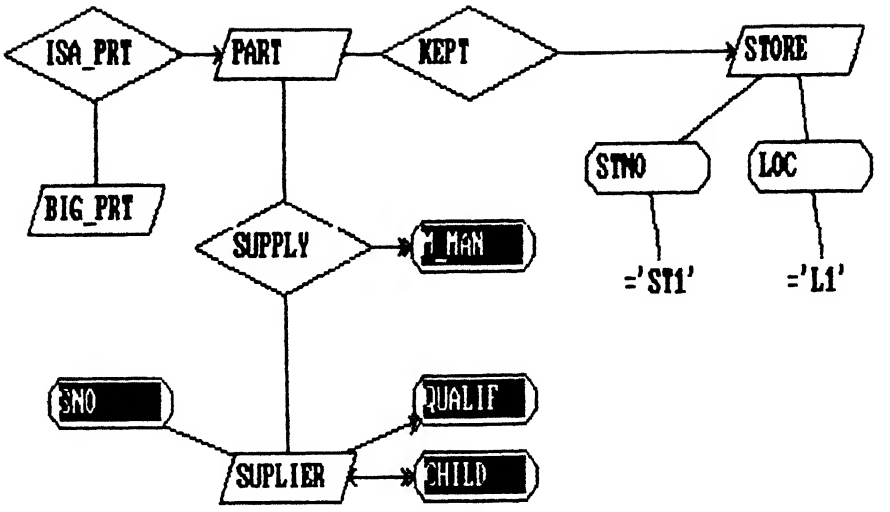


Fig 4.20 Query: Show all supplier numbers, their children and qualifications, and the middle man associated with the supply of the big parts kept in a store ST1 located at L1.

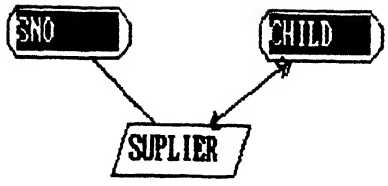


Fig 4.21(a) Query representing the table SUPLIER$1.

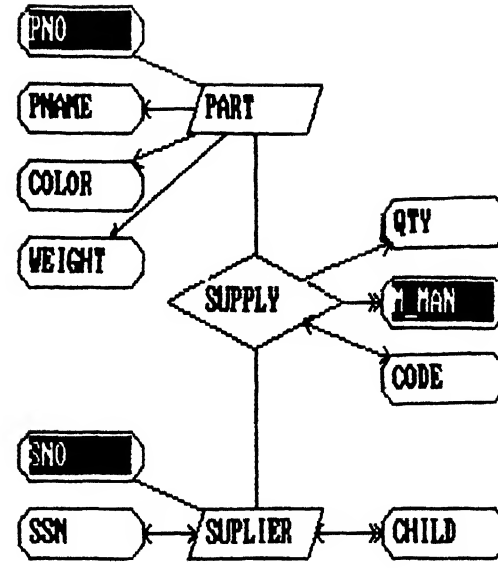


Fig 4.21(b) Query representing the table SUPPLY$1.

Fig 4.22 Query representing the equivalent of union in relational algebra.

Fig 4.23 Query representing the equivalent of minus in relation algebra.

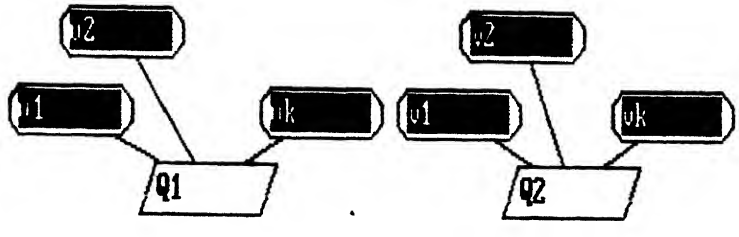


Fig 4.24 Query representing the equivalent of cross product in relational algebra.

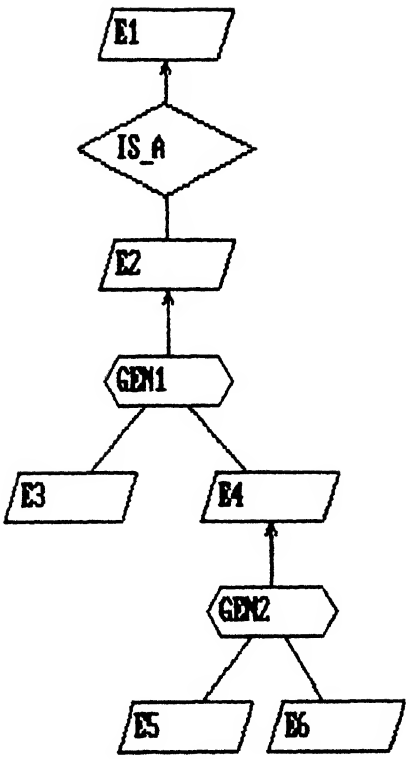


Fig 4.25 Schema showing root entity definition in the generalization hierarchy. E2 is a root entity.

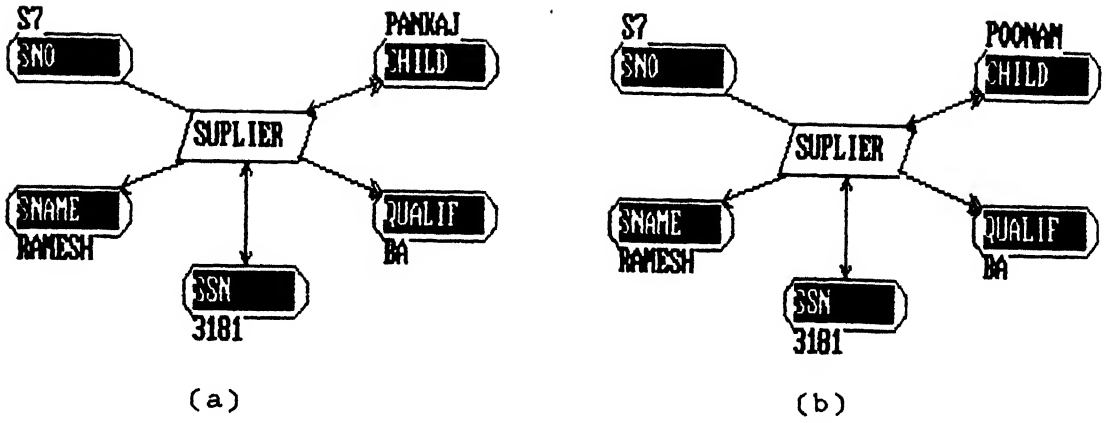


Fig 4.26 Browsing of entity "SUPLIER".

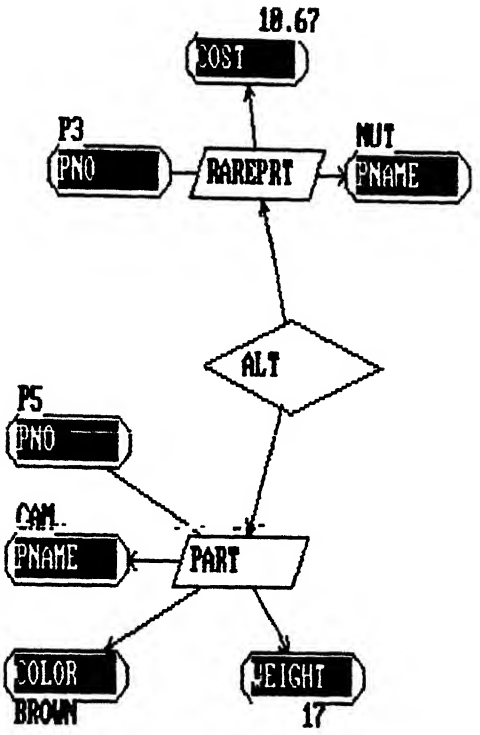


Fig 4.27 Browsing relationship "ALT".

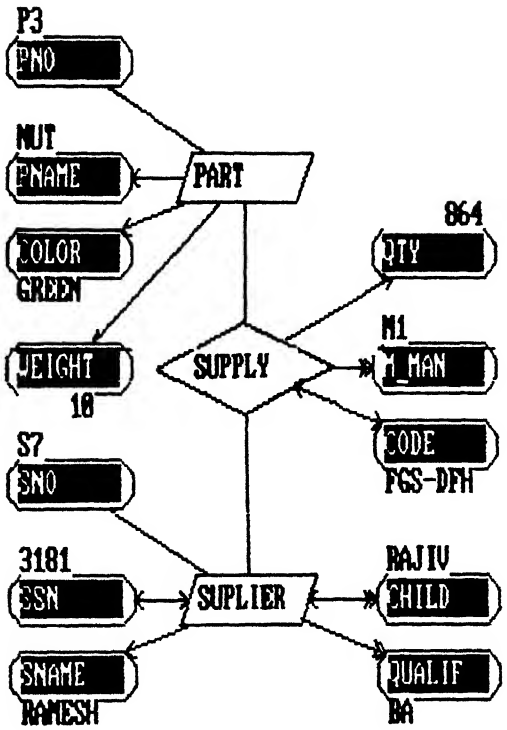


Fig 4.28 Browsing relationship "SUPPLY".

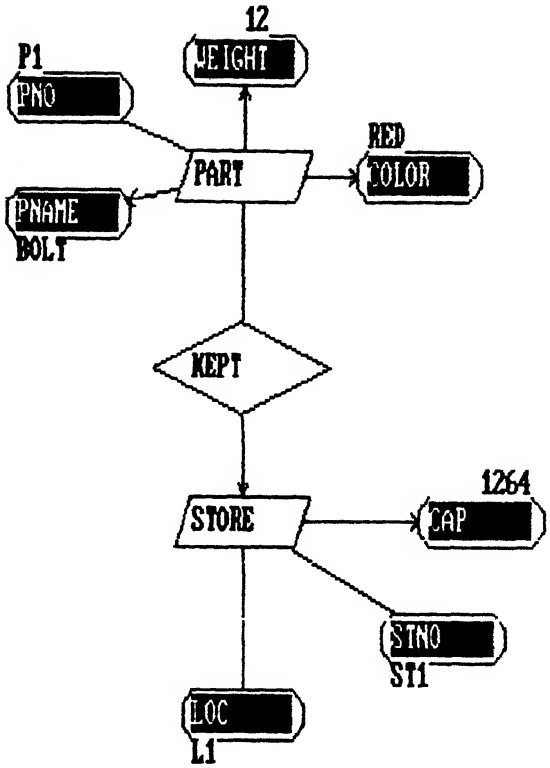


Fig 4.29 Browsing relationship "KEPT".

Chapter 5

Implementation Details

The implementation of the GQL is done in 'C' language under MS-DOS operating system. The program is divided into the following modules:

1) Graphics editor to construct schema/Query.
2) Module to convert EER diagram to relations.
3) Module to convert GQL queries to SQL queries.
4) Module for Graphical updating of data.
5) Module for Browsing.
6) Module to interface ORACLE to execute SQL queries.

Few of the important data structures are discussed below.

1) structure of an attribute

```
struct attribute {
    char name[11]; /* Name of the attribute */
    char type[16]; /* Type of the attribute */
    char map;      /* Mapping of the attribute */
    struct entity *ent; /* pointer to the entity it belongs */
    struct relation *rel; /* pointer to the relationship it
                             belongs */
    /* an attribute can belong to either an entity or a
       relationship. When pointer to entity is NULL then it
       shows that attribute belongs to relationship and vice
       versa. */
    struct column *col; /* pointer to the corresponding column
```

```
                        in the relation. In case it is a key
                        attribute it points to the column of
                        E-1 type relation */
    struct gelement *gele; /* pointer to graphics structure */
}
```

2) structure of an entity

```
struct entity {
    char name[11]; /* name of the entity */
    char type;     /* type of entity i.e. strong, subset of
                      generalization/specialization hierarchy */
    struct attribute *att[]; /* Array of pointers to the
                                attribute of an entity */
    struct relation *rel[]; /* Array of pointers to relationship
                               to whom the entity is associated */
    struct gelement *gele; /* pointer to graphics structure */
}
```

3) structure of a relationship

```
struct relationship {
    char name[11]; /* name of the relationship */
    char type;     /* type of relationship */
    struct attribute *att[]; /* Array of pointers to the
                                attribute of an relationship */
    struct entity *ent[]; /* Array of pointers to the entities
                             associated with the relationship */
    struct gelement *gele; /* pointer to graphics structure */
}
```

```
4) structure of a graphics element
struct gelement {
    int x, y; /* coordinates of the object on the screen */
    int color; /* color of the object */
    int num; /* type of the object it points to */
    *ptr; /* this is a union pointer to different objects */
}
5) structure of a column of a table
struct column {
    char name; /* name of the column */
    char type; /* type of the column */
    struct table *tab; /* pointer to the table it belongs */
    struct attribute *att; /* pointer to the corresponding
                              attribute */
}
6) structure of a table
struct table {
    char name[24]; /* name of the table */
    struct column *col[]; /* Array of pointer to the columns
                             of the table */
    struct entity *ent; /* pointer to the entity */
    struct relation *rel; /* pointer to the relationship */
    /* when pointer to entity is NULL then the table
       corresponds to the relationship and vice versa */
}
```

Chapter 6

Conclusions and Future work

If the database schema is very large, then the number of relations (table) used for storing the data also become very large. It then becomes difficult to remember the relations and use them for querying. The graphical query language is a very powerful tool for manipulating large databases at conceptual level. Most of the queries can be easily formulated using GQL. GQL can also be easily used by the naive users without much prior knowledge of databases. The other advantage of using conceptual databases is that the choice of the physical data model can be decided according to the the nature of the database schema, but the query language can be same for all.

Many graphical query languages exist over the Entity Relationship model. But they do not incorporate many features of the EER model. We have tried to incorporate many such features in it. Category abstraction and different types of attribute mappings are some such features that have been incorporated. In our GQL data consistency checks are done automatically.

Even though GQL is as powerful as the relational algebra, it is difficult to construct certain queries specially those requiring multiple levels of nesting. Further since queries in GQL are translated using the natural join of all the entities and

relationships participating in the query, the queries requiring θ-join become difficult to represent.

The GQL defined can easily be extended to incorporate "functions". Queries like "show details of the heaviest part" can then be easily represented. Constraints like "a supplier should not have more than two children" can not be currently represented. Facilities can be provided to represent such and other additional constraints. The GQL can also be extended to incorporate "aggregation" and multiple keys for an entity.

We feel that the limitations of this Graphical Query Language can be overcome without much difficulty. We also feel that after overcoming these limitations and incorporating the above suggestions, this query language will become a very powerful tool in database systems.

Chapter 7

References

[1] Chen P. P., "The Entity-Relationship Model: Towards a unified view of data", ACM TODS 1 (1976), 9-36.

[2] Czejdo B., "Semantics of Update operation for an Extended Entity-Relationship Model", ACM (1988), 178-188.

[3] Chan E. P. F., "A Graphical database design AID using the Entity-Relationship model", Entity Relationship approach to system analysis and Design, Chen P. P. (Ed.), North Holland Publishing Company, (1980), 295-310.

[4] Zhang Z. Q., "A Graphical Query Language for Entity-Relationship databases", Entity Relationship approach to software engineering, Elsevier Science Publishers B. V. (North Holland) 1983, 441-448.

[5] Hoeve F. V. "An object oriented approach to Application Generations", Software practice and experience, vol. 17(9), (1987), 623-645.

[6] Pahwa A., "Automatic database Navigation: Towards a high-level user interface", Entity Relationship Approach, IEEE CS press, North Holland, (1985).

[7] Azmoodeh M., "GQL, A Graphical Query Language for semantic databases".

[8] Elmasri R. A., "A Graphical Query Facility for Entity Relationship databases", Entity Relationship Approach, IEEE press, North Holland, (1985).

[9] Atzeni P., "INCOD: A system for conceptual design of data and transaction in the Entity Relationship model", Entity-Relationship Approach to Information Modeling and Analysis, Chen P. P. (Ed.), Amsterdon, The Netherlands: North Holland, (1983), 379-414.

[10] Smith J. M., "Database Abstraction: Aggregation and Generalization", ACM TODS vol.2, (1976), 105-133.

[11] Theorey T. J., "A Logical Design Methodology for relational databases using the Extended Entity Relationship Model", ACM Computing Surveys, vol.18(2), (June 1986).

[12] Ling T. W., "A Normal Form for Entity-Relationship diagram", Entity-Relationship Approach, IEEE CS press, North Holland, (1985), 24-35.

[13] Nixon B., "Implementation of a compiler for a semantic data model: Experiences with Taxis", Sigmod Record, vol. 16(3), Dec 1987, 118-131.

[14] Lyngback P., "Mapping a semantic database model to the Relational Model", Sigmod record, vol. 16(3), Dec 1987, 132-142.

[15] "Semantic Nets", SIGMOD 81.

[16] Kunni T., Visual database system, IFIP, North Holland.

[17] Jagannathan D., "SIM: A Database System Based on the Semantic Data Model", Sigmod record vol. 17(3),(sept 1988), 46-55.

[18] "Nested Datalog", SIGMOD 81.

Appendix

Manual

This package provides a Graphical Query Language facility over an Extended Entity Relationship diagram. This works over the "ORACLE RDBMS". It requires the ORACLE RDBMS to be installed on to the IBM PC/AT (or Compatible) with an Enhanced Graphics adapter(EGA). Before running this package you should already have a login account on the ORACLE and oracle should be loaded before. To start this package type

C>EER-BASE <—┘

It will then ask for your login name and the password. After successfully logging in ORACLE, you will see the screen as shown in fig 1. The screen is divided into three area.

(i) Schema Area: Schema area shows you the part of the schema which can be viewed through the current window position.

(ii) Menu Area: Menu area gives the definition of the function keys which are currently valid. Menu area is also used to take some input information from the user. It also displays the current cursor coordinates.

(iii) Message area: Message area displays the different messages to help or provide necessary information to the user. Whenever an error or an important message is displayed, it is followed by a beep.

The screen layout is same for all the options (i.e. SCHEMA,

QUERY, UPDATE, BROWSE) shown in Fig. 1. A Graphics Editor is provided for SCHEMA, QUERY, BROWSE, and UPDATE options. The EER diagram can be contained in the six pages (2 in x-direction and 3 in y-direction). Only the part of the EER diagram which can be viewed through the scheme window is shown.

Following are the commands available for Graphics editor.

Pg Up: Moves the schema window up by half page length.

Pg Dn: Moves the schema window down by half page length.

Home : Moves the schema window left by half page width.

End : Moves the schema window right by half page width.

Left Arrow : Moves the cursor by one position left.

Right Arrow: Moves the cursor by one position right.

Up Arrow : Moves the cursor by one position Up.

Down Arrow : Moves the cursor by one position Down.

CNTL Left : Moves the cursor by eight positions left.

CNTL Right : Moves the cursor by eight positions right.

CNTL Down : Moves the cursor by four positions down.

CNTL Up : Moves the cursor by four positions Up.

+ : Marks the current position of the cursor as default position.

* : Moves the cursor to the default position.

- : Marks the current position of the cursor as default position and moves the cursor to the previous default position.

/ : Searches for the string forward, starting from the current position of the cursor. If no search string is specified

then, it searches for the previous string.

\ : Searches for the string backward, starting from the current position of the cursor. If no search string is specified then, it searches for the previous string.

a : Searches an Attribute (Forward).

A : Searches an Attribute (Backward).

e : Searches an Entity (Forward).

E : Searches an Entity (Backward).

r : Searches a Relationship (Forward).

R : Searches a Relationship (Backward).

l or L: Loads the schema.

p or P: Prints the schema or query.

It asks the user to give the file name in which the bitmap of the schema/query should be stored. Then this file can be printed by using the "print.exe" file.

m or M: This command is used to move an object (Relationship, Entity, or Attribute). The object is selected by the current

position of the cursor. Now this object can be moved by using the arrow keys. All the connection to the other objects are automatically refreshed. Press ESC key to end moving session.

An attribute can not be moved if it is participating in the condition of the graphical query.

v or V: This command is used to verify the validity of a

schema, query, or browse.

Now let us discuss each option of Fig. 1.

1) **SCHEMA**: This option is used to create and define a database schema in the form of EER diagram. Let us discuss the definition of each Function Key.

F1 : Add Entity

This is used to add a new entity to the EER diagram. On pressing F1, the user is asked to give the name and type of the entity. Type of the entity is selected by pressing the "Up Arrow" or "Down Arrow" key. The '*' shows the type of the entity currently selected. At any time the operation can be aborted by pressing ESC key. After selecting the "type" an entity is displayed at the current cursor position. The entity can be moved by using the "move" command.

F2 : Add Relation

This is used to add a new relationship to the EER diagram. The user is asked to enter the name and select the type of the relationship.

F3 : Add Attribute

This is used to add a new attribute to a entity or a relationship. First, the user is asked to give the attribute name and then the domain of the attribute. Valid domains are CHAR and NUMBER. After this, the user is asked to select the type of the attribute. Now a message is displayed to select the entity or relationship to which this belongs. Selection is done by moving

the cursor over an entity or a relationship and then pressing RETURN key.

F4 : Add ER Link

This is used to add a link between a relationship and an entity. User is asked to first select the Relationship and then the Entity. After this, the user is asked to select the connectivity of the link.

F5 : Delete

This is used to delete the object pointed by the cursor. In case of Entity/Relationship all the attributes belonging to that Entity/Relationship are also deleted. In case of a Relationship all the links to the Entities associated with it are also deleted.

F6 : Delete link

This is used to delete the connectivity between a relationship and a entity.

F7 : Modify

This is used to modify the name of an object pointed by the cursor.

F8 : Save Schema

This is used to save the EER diagram. A file "SCHEMA.DAT" is created and the information is saved in that file. User should always keep a separate copy of this file.

F9 : Def Schema

This is used only when the designing and making of an EER diagram is over. At this point first EER diagram is verified for

the validity. After this equivalent relations are created. When schema is defined then schema is locked for further modification. i.e. all Function keys are invalid except F5 and F10. F5 can be used to delete only a named query.

F10: Drop Schema

This is used to drop the defined schema. Two times user is warned before dropping the schema. Dropping the scheme means that all the relations and their contents are lost.

2) QUERY

This option is used to make Graphical queries and then to execute them. The Function keys are defined as discussed below.

F1 : Mark

This is used to mark the attributes for displaying query results. This can also be used as to unmark some attributes. When marking session is over then press ESC key.

F2 : Set Cond

This is used to set a condition over the attributes. It is also used to add a condition relating two named queries.

F3 : Duplicate

This is used to duplicate an entity or a relationship. When a relationship is duplicated then, its associated links to entities should be specified by using F4 function key.

F4 : Compose

This is used to add link between a relationship or an entity. This is normally used after the duplication of a

relationship.

F5 : Delete

This is same as the delete operation on a schema. This is normally used to delete the relationships which are not participating in the query.

F6 : Name Query

This is used to name a valid query. User is first asked to select the place in the EER diagram for the query, followed by the name of the query. Then one by one position and new name for all the marked attributes is asked.

F7 : Show attribute

This is used to display an attribute inherited from the parent entity. It can also be used to undelete a deleted attribute.

F8 : Run

This is used to run the graphical query formulated by the user.

F9 : Del ER link

This is same as the "Del Link" of the SCHEMA menu. This is used to delete the link between Generalization relation and the subset entity.

F10: Mark Rel

This is used to mark the relationship participating in the query. All the unmarked relationships will be automatically deleted from the diagram.

3) UPDATE

This option allows the user to perform update operations on the EER diagram. The update operation is performed by selecting the object and the kind of operation to be performed. User is then asked to enter the relevant values. Execution of the update operation is shown as the operation progresses. The function keys are defined as discussed below.

F1 : Insert

This option is used to insert an instance into an entity or a relationship. To perform this option, move cursor over an entity or a relationship and then press F1 key. In case of an entity, user is asked to enter values for all the attributes of an entity. Values of key attributes are mandatory while for descriptor attributes they are optional. If more instances of a multivalued attribute are required then enter the value of the key attributes and that of multivalued attributes. In the case of a relationship, user is asked to enter the values of all the attributes of entities associated with the relationship and the attributes belonging to the relationship.

Press ESC to exit insert mode.

F2 : Delete

This option is used to delete an instance from an entity or a relationship. User is asked to enter the value of the key attributes of that object.

F3 : Modify

This option is used to modify the value of attributes. User

is asked to enter the value of the key attributes of the object to which the attribute belongs. In case of multivalued attributes, old value of that attribute is also asked.

4) BROWSE

This option allows the user to examine the structure of database.

F1 : Browse Rel

This option allows the user to browse a relationship. The key is pressed against the relationship to be browsed. All attributes present are selected for display. The first instance is retrieved from the database and displayed.

F2 : Next

This option is used to retrieve the next instance for browsing the relationship. The key should be pressed against an attribute.

F3 : Prev

This option is used to retrieve the previous instance for browsing the relationship.

F4 : Browse Ent

This option allows a user to browse an entity.

F5 : Next

This option is used to retrieve the next instance for browsing an entity. This key can also be used to browse an entity while browsing a relationship.

F6 : Prev

This option is used to retrieve the previous instance while browsing an entity. This key can also be used to browse an entity while browsing a relationship

5) SQL

This option allows the user to execute any SQL command. The command must be terminated by ";". For example

```
SQL> select * from part
SQL> where color = 'RED';
```

This query will display all red parts.

6) Exit

Exit the program.

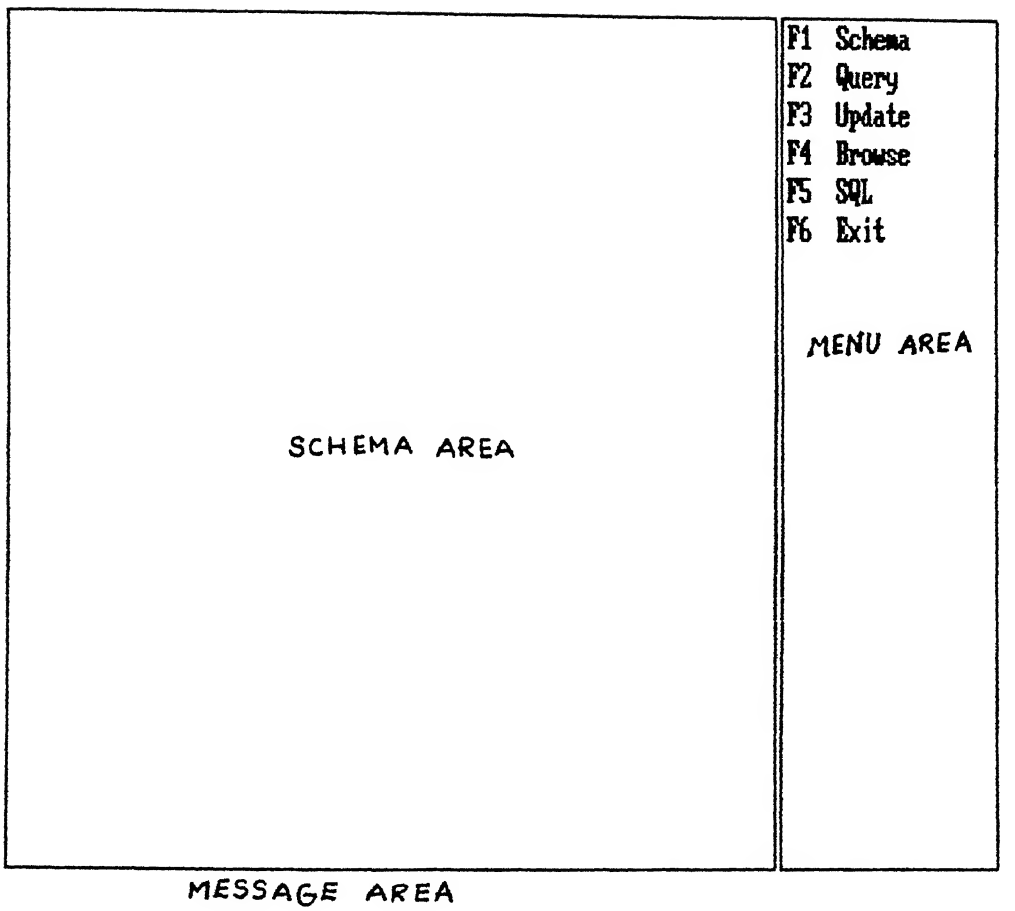


FIG. 1 SCREEN LAYOUT